A CHATTER-FREE POCKETING ROUTINE FOR ANY TWO-AND-A-HALF-DIMENSIONAL POCKET WITH ISLANDS

BY

E-HSUNG CHENG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1992

UNIVERSITY OF FLORIDA LIBRARIES

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Dr. Jiri Tlusty, his advisor in this project. Without his support and guidance, this study would not have been possible. He would also like to thank Dr. Kevin Smith for his invaluable advice to the project. Appreciation is also extended to his committee members, Dr. John Schueller, Dr. Donald Dareing, and Dr. Senser Yeralan, for reviewing the dissertation. Gratitude is also extended to Mr. Wei Guo Zhang for his assistance in the laboratory work and to Mr. Chun-Yin Wu for his initial suggestion about the program structure. Finally, the author would like to express his sincere gratitude to his parents, Chau-Lu and Jan-Jan, and his wife, Jaw-Fen Wang, for their encouragement, understanding, and financial support. To them the author offers his heartfelt thanks.

TABLE OF CONTENTS

page

ACKNOWLEDGEMENTS ii

ABSTRACT v

CHAPTERS

1 INTRODUCTION 1

1.1 Introduction to Pocketing Routines 1
1.2 Offsetting Algorithm 4
1.3 Some Considerations of the New Pocketing Routine 6
1.3.1 Chatter Consideration 7
1.3.2 Corner Consideration 12
1.4 Dynamic Spiral Pocketing Routines 12

2 OPTIMIZATION PROCESS 20

3 PROGRAM STRUCTURE 27

3.1 Data Representation of a Segment 27
3.2 Linked list Representation 29
3.3 Tree Representation of a Pocket and its Spiral Tool Paths 35
3.4 Program Implementation of a Binary Tree 38
3.5 Island Representation 40
3.6 Tree Traversal 44

4 SOME SUPPORTING SUBROUTINES 49

4.1 Classifying Inside-Outside Relation of a Point and a Closed Loop 49
4.2 Intersection of Two Segments 55

5 FINDING THE TOOL PATH OF A POCKET 62

5.1 Finding Offset Loops of Equal Offset Distance 63
5.1.1 Finding the Offset of Each Segment on the Original Loop 63
5.1.2 Linking Pair by Pair the Offset Segments Whose Original Segments Are Adjacent 65

5.1.2.A Case 1: The Angle Is Greater Than 180° 65
5.1.2.B Case 2: The Angle Is Lesser Than 180° 66
5.1.3 Splitting the Gross Offset Loop into Several Sub-loops 68
5.2 Finding Offset Loops of Unequal Offset Distances 73
5.2.1 Finding the Offset of Each Segment on the Original Loop 73
5.2.2 Connecting Each Pair of Neighboring Offsets to Form a Gross Offset Loop 76
5.3 Tool Path of a Pocket with Islands 80
5.4 Loop Termination Conditions 88
5.5 Connection between Larger Loops and their Sub-loops 90
5.5.1 Two Cases of Connection between Loops 91
5.5.2 Retract and Approach 96
5.5.3 Change the Starting Segments of Loops 97
5.5.4 Connection of Loops While Islands exist ... 98

6 INSTALLATION PROCESSES 101

6.1 Installation of the New Pocketing Routine in CATIA 101
6.2 APT and Machine Codes 106

7 EXAMPLES OF TOOL PATHS 110

8 CONCLUSIONS AND RECOMMENDATIONS 120

APPENDIX

A THE CHARACTERISTIC OF A CUTTING TOOL 123

BIBLIOGRAPHY 126

BIOGRAPHICAL SKETCH 128

Abstract of Dissertation Presented to the Graduate School
Of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A CHATTER-FREE POCKETING ROUTINE FOR ANY
TWO-AND-A-HALF-DIMENSIONAL POCKET
WITH ISLANDS

BY

E-HSUNG CHENG

MAY 1992

Chairman: Dr. Jiri Tlusty
Major Department: Mechanical Engineering

Chatter has always been a significant problem in high speed, high power machining. For the purpose of chatter avoidance, the primary parameters are the axial and the radial depths of cut which are decisive in the tool path optimization. The acceptable (chatter-free) axial and radial depths of cut change depending on the dynamics of the machine tool, the direction of cut, and the mode of milling (up or down milling). The relationship of the above parameters has been incorporated to form a database by other collaborators in the Machine Tool Laboratory at the University of Florida.

The pockets used in this work may have complex 2½-dimensional geometries and may include multiple islands. Two main approaches have been developed to achieve the chatter-

free routine. At first the optimum radial and axial immersions would be determined according to the database and the pocket geometries. Secondly, the tool path is arranged using the optimum immersions to achieve the best metal removal rate without the occurrence of chatter. In the optimization process, different radial immersions may be selected for different machining directions. This feature is unique since other pocketing routines can only choose the same immersion for all machining directions. Therefore, the machine would not use all of its power capabilities, since the smallest immersion has to be selected if those routines are used.

This work also outlines a complex general tree data structure, the implementation of geometric modelling techniques into programming, and the method to incorporate the new routine into CATIA, a widely used CAD/CAM package. Finally, a processor is provided to established a link between CATIA and an NC machine.

Several examples are used to illustrate the resulting tool paths from the new routine, and comparisons are made between the actual parts machined by tool paths resulting from the new routine and from the original pocketing routine from CATIA.

This work presents a new type of pocketing routine which would reduce the manufacturing time and increase machining quality and tool life.

CHAPTER 1

INTRODUCTION

1.1 Introduction to Pocketing Routines

A common procedure in a milling operation is to remove the material within a given boundary and to create a cavity, also known as a pocket. The pocket can have any shape along its boundary. Machining aircraft panels and molding dies are two common examples of pocket use. Pocketing routines have been implemented in many numerical control (NC) packages, and may be grouped into three broad categories:

(1) Spiral out: In such a routine, the tool is first sunk in near the center of the pocket, and then traverses a path spirally outward. The last path will create the periphery of the pocket.

(2) Zig-Zag: The starting point is at a distance of the tool radius plus the finishing cut immersion from a chosen point on the periphery. The tool is first sunk at the starting point, and then traverses back and forth, cutting in both directions across the pocket, and finally finishes on the opposite side of the pocket from the starting point. The final path is the finishing path which is used to create the periphery of the pocket.

(3) Linear: This routine is similar to the zig-zag routine, except that the cutting is performed only in one direction, and on the return path the cutter is withdrawn and travels in air. In order to avoid the necessity of sinking at the beginning of each pass, the periphery of the pocket is first machined in a slotting operation, before the linear motion begins.

Among the above three operations, the spiral pocketing operation is the most complicated but also the most efficient [1][2], and thus is the primary subject of this research. Although the spiral pocketing routine has been widely used, none of the commercial packages have given any description about their algorithms, and few publications about the pocketing routines are available. These publications have not provided a good way to deal with a pocket with many islands. Furthermore, none of the routines in the commercial NC packages take into account the problems of chatter and internal corners which are crucial in the high speed high power machining operation. These problems will be addressed in detail later in this chapter. This research attempts to solve the above problems and also provide a robust algorithm in dealing with a pocket of complex 2-D geometries with or without multiple islands.

Few publications have been presented about pocketing routines. Persson's [3] work describes an efficient two-stage procedure for a pocket bounded by line segments and circular

arcs. This pocket was machined in a spiral fashion. He first partitioned the entire pocket into several independent subareas each belonging to one pocket side, and then the cutter paths were determined such that each path was found in a subarea at a time. The subarea was partitioned by calculating the curves (lines, circular arcs, ellipses, parabolas, or hyperbolas) equidistant between each pair of adjacent boundaries of the pocket. However, when dealing with unequal radial immersion at each pocket boundary (this will later be shown to be desirable), this kind of partition will not work. Additionally, his pocketing procedure does not permit internal island avoidance. Nevertheless, he mentions that lines and arcs alone are not always sufficient for representing airplane parts. These complex representations include tabcyls, which are curves defined by point sets. He developed a prepocket procedure which approximated tabcyls with arcs and lines, before the normal pocket calculations were started.

Another publication about pocketing routines is the one by Suh and Lee [4]. Their work described the machining of a pocket spirally. The pocket may be bounded by many curves, including lines, arcs, ellipses, and arbitrary free curves. The pocketing routine was capable of island avoidance, but only one island was allowed. Additionally the bottom of the pocket to be machined may be any sculptured surface.

Their procedure started from a three-dimensional (3-D) model generated by a geometric modeler developed in their laboratory, which was a nonuniform rational B-spline-based boundary representation modeler. Once the shape of the pocket had been modelled, the offset curves and the associated CL data were computed at every offsetting step. At each step, one or more loops would be generated.

However, their routine was useful only for the finishing cut of the pocket which emphasizes the precision and continuity of the tool movements on the sculptured surface, while in a roughing cut, the metal removal rate is a more important issue. Also, in their routine only one island was allowed, which implied that their data structure would need more modification. Furthermore, they did not describe the procedure to connect a loop and its offsetting loops. The connection will become quite complicated in the case of machining a complex pocket enclosing many islands.

As in the commercial pocketing routines, both Persson's and Suh's works were concentrated on the equal distance offset principle, and did not consider the problems associated with internal corners.

1.2 Offsetting Algorithm

An offset is a curve with equal distance to a given curve. The offset technique can be applied in several fields, including NC pocket machining, VLSI circuit design, finite

element mesh generation, and robot path planning. In conventional spiral type pocketing routines, tool paths are formed as loops. The sub-loops are the offset of their corresponding larger loops. The offset technique has been discussed in several papers. Tiller and Hanson [5] discussed the offsets of two-dimensional profiles. They used the subdivision method to find the offset of a rational B-spline curve. But their method runs into trouble when the offset curve intersects itself. Pham [6] provided a method to interactively find the offset of a uniform cubic B-spline by using a set of control knots. He discussed only the problem of offsetting a curve, and not a loop, for in a loop the problem is more complicated. Hoschek's work [7] discussed the problems involved in generating the offset of a cubic spline. Coquillart's [8] work discussed the offset of a B-spline in both 3-D and 2-D. But his work was limited to finding the offset of smooth curves without discontinuities. Moreover, self-loops, which are loops within a larger closed loop, are not allowed in offset curves.

The above papers concentrated their efforts on finding the offsets of splines, either B-splines or cubic splines. The offset will be a spline of the same kind. These efforts are reasonable, since the spline is a powerful tool for the designer and can be used to approximate any desired curve.

The offset of a line segment is another line segment, and the offset of an arc is another arc. In some cases, a pocket

is constructed by not only line segments and arcs, but also by ellipses, hyperbolas, and parabolas. The exact offset of an ellipse (hyperbola, or parabola) is a fourth order curve which can still be found, but finding the exact offset of this fourth order curve becomes very complicated if it even exists. It is easier to use a spline to approximate the offset of this fourth order curve. As a result, it is necessary to have a method to find the offset of splines. In other words, it may be said that if a pocket is bounded by curves beyond line segments and arcs, the above papers might provide a method to solve the offset problems. However, the kind of offset required in this research is not of the equal distance form. The offset distance is variable according to the direction of each pocket edge and is determined by optimization results. None of the above papers has ever mentioned anything about variable distance offsets. The problems involved are of a new nature. Thus, in this research the types of segments defining any pocket boundaries are limited to line segments and circular arcs.

1.3 Some Considerations of the New Pocketing Routine

Assuming that the criteria for a satisfactory cut is the avoidance of chatter, it is necessary to explain the meaning of chatter, and why, for a given axial depth of cut, the radial depth of cut may be different for each direction of machining.

1.3.1 Chatter Consideration

The self-excited vibration in machining, also known as chatter [9], is an important factor for controlling the quality of the cutting operation. Chatter is recognized by characteristic noise, by chatter marks, and by the undulated or dissected chip. The mechanisms producing chatter are "mode coupling" and "regeneration of waviness." The latter plays a more significant role.

The limit of stability for regenerative chatter for a single degree of freedom system is shown in Figure 1.1. It is obvious that the projection F $\cos(\beta-\alpha)$ of the cutting force excites the mode in the X direction and the projection X $\cos(\alpha)$ of the vibration in the Y direction will modulate the chip thickness. The oriented transfer function is [9]

$$G(\omega) = \frac{Y(\omega)}{F(\omega)} = \cos(\beta-\alpha)\cos(\alpha)\frac{X(\omega)}{F(\omega)} = \mu G_d, \qquad (1.1)$$

where $\mu = \cos(\beta-\alpha)\cos(\alpha)$ is the directional factor, and G_d represents the direct transfer function. In the preceding pass, the vibration is given by

$$y_0 = Y_0 \sin(\omega t) \qquad (1.2)$$

and, in the current pass, the vibration is

$$y = Y \sin(\omega t + 2\pi N + \epsilon) = Y \sin(\omega t + \epsilon) \qquad (1.3)$$

The cutting force f is

$$f = f_m + f_v \qquad (1.4)$$

where f_m is the mean cutting force, and f_v is the variable

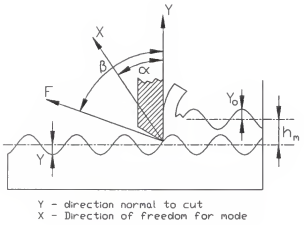


Figure 1.1 Mechnism of "regeneration of waviness"

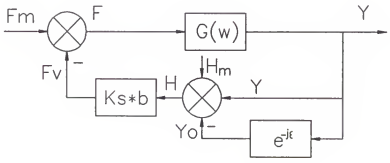


Figure 1.2 Block diagram of chatter mechnism

component. Their values are

$$f_m = k_s\, b\, h_m, \tag{1.5}$$

$$f_v = k_s\, b\, (y_0 - y), \tag{1.6}$$

where K_s is the cutting force coefficient, and b, the depth of cut, is a positive real number.

The above equations can be represented by a block diagram in Figure 1.2. At the limit of stability, any vibration would remain constant without decaying or increasing. Thus, the absolute values of amplitudes for two subsequent vibrations are equal.

$$| Y | = | Y_0 | \tag{1.7}$$

According to the Nyquist criterion, the limit happens when the open-loop transfer function of Figure 1.2 has the value -1

$$K_s\, b_{lim}\, G(\omega)\,(1 - e^{-j\epsilon}) = -1,$$

$$b_{lim} = \frac{-1}{K_s\, G(\omega)\,(1 - e^{-1\,\epsilon})} \tag{1.8}$$

since b_{lim} is always real and K_s is chosen to be positive real. The above equation can be justified only if $G(1 - e^{-j\epsilon})$ is real. This condition can be satisfied if

$$G\,(1 - e^{-j\epsilon}) = 2\, Re(G) \tag{1.9}$$

Thus,

$$b_{lim} = \frac{-1}{2\, K_s\, Re(G(\omega))} = \frac{-1}{2\, \mu\, K_s\, Re(G_d)} \tag{1.10}$$

The single degree of freedom system can only be used to

explain the mechanism of chatter vibration, and is not suitable for the real machining process. For a two degree of freedom system, the equation for b_{lim} becomes

$$b_{lim} = \frac{-1}{2\,K_s\,(\mu_x\,Re(G_x) + \mu_y\,Re(G_y))_{min}} \qquad (1.11)$$

where

$Re(G_x)$: real part of the transfer function measured in the X direction of the feed motion,

$Re(G_y)$: real part of the transfer function measured in the Y direction perpendicular to the feed motion,

μ_x: directional orientation factor for the X dynamics,

μ_y: directional orientation factor for the Y dynamics.

If only the tool were involved in determining the transfer function at the tool tip position, then the transfer functions G_x and G_y in the X and Y directions would be the same. However, in most practical cases the machine structure must be considered, and thus the transfer functions G_x and G_y may be significantly different. The method of determining μ_x and μ_y was explained by Tlusty [9]. It is known that the values of μ_x and μ_y are determined by the machining direction and the mode of milling which is either up or down milling, and can be quite different. Adding the factors of changeable μ_x and μ_y, and of the difference between G_x and G_y, the minimum value of $\mu_x Re(G_x) + \mu_y Re(G_y)$ may be considerably different for different cutting conditions. Accordingly, from Equation 1.11, the value of b_{lim} will be different depending on whether it is up

or down milling and on the machining direction.

None of the commercial NC packages have offered the option of selecting different radial immersions for different cutting directions. All that is available is a constant immersion for all directions. Therefore, to ensure a stable milling operation, it is necessary to choose an immersion based on the worst direction and immersion. Thus, in cutting in the more stable directions the machine does not use its full capability. Even if a pocketing routine offers the user the option to change the radial immersion according to the direction and the mode of milling, in the existing routines there is no guideline for selecting the parameters. The programmer is left to his own experience to select the radial and axial immersions which will yield an acceptable quality.

A database of quality parameters, which is the pre-work of this research, is proposed. The database, developed by experiments or a time domain simulation program in the Machine Tool Laboratory at the University of Florida, will include a representative selection of tools. Thus, while a shape of the pocket, and the tool used to machine it are determined, the new pocketing routine will first determine the axial depth of cut and the radial immersion for each side of the pocket by an optimization technique such that the best metal removal rate is achieved. The method of optimization is the topic of Chapter 2.

1.3.2 Corner Consideration

Another problem neglected by the commercial spiral-out pocketing routines is the corner consideration. As the tool approaches an internal corner, the radial immersion starts to increase until it is on the corner, and then the immersion returns to normal.

Figure 1.3 shows an example of a half immersion cutting operation. At point 1 and point 3, the immersion is half, and the cutting angle, which is the angle of engagement between the work piece and the cutter, is 90°. But at point 2 there is a slotting operation and the cutting angle is 180°. As is shown above the radial immersion has a strong affect on the stability of the operation. The increase in immersion at the corner may be sufficient to cause chatter.

Tsai and Takata [10] tried to add additional tool path segments (arcs) for machining corners so as to keep the engagement angle below a certain value. However, their method was complex and the machining became inefficient.

1.4 Dynamic Spiral-in Pocketing Routine

Kshonze [1] and Tlusty [2] have provided a guideline for the dynamic routine which had the following two stages:

1. Performing the slotting around the inside perimeter of the pocket;
2. Spirally cutting from the periphery of the pocket toward the center.

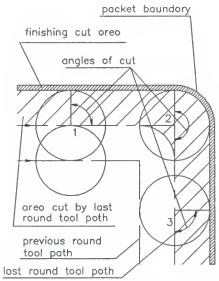


Figure 1.3 Spiral-out tool path at an inside corner

For the slotting of the periphery, the axial depth of cut was chosen according to the worst direction from the data base. Since the other directions were allowed to cut deeper than that, the slotting was stable. The slotting might be done by several layers according to the depth of the pocket and the maximum allowable axial depth of cut. For the remainder of the pocket, the radial and axial depths of cut would be determined by using an optimization process to achieve the best metal removal rate. Using the optimization process is the first advantage of the dynamic spiral-in pocketing routine.

Such a tool path is shown in Figure 1.4 for a simple square pocket. First the initial slotting is done in two stable axial passes (2,3,4,5,2 and 6,7,8,9,6). After that, assuming the X direction of feed to be the less stable option,

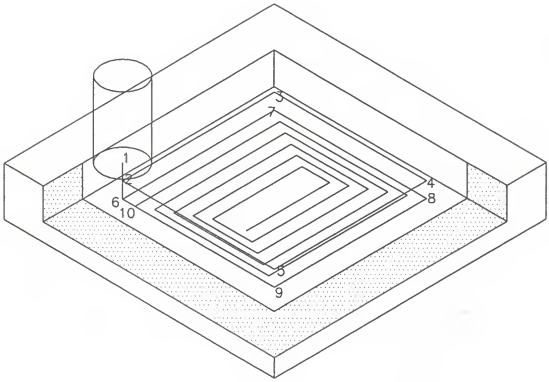


Figure 1.4 The dynamic spiral-in tool path for a square pocket

for a full axial depth of cut, a 1/5 radial immersion is chosen. In the Y direction of feed, a 2/5 radial immersion is chosen. The remaining pocket is machined in one pass at full depth from point 10 in a spiral motion toward the center of the pocket.

The second advantage of the dynamic spiral-in routine is typical for spiral-in routines for they avoid the increase of the cutting angle at the inside corners (refer to Section 1.3.2). Figure 1.5 shows that after the initial slotting, less material is cut at the inside corner area than at the non-corner areas.

However, a pocket may also include outside corners, although in a closed area (pocket) the number of outside corners is always less than the number of inside corners. For outside corners, the tool path proposed is to cut around it during the initial slotting using an arc whose radius is the same as the tool radius. After the first slot, there will be no outside corners left. In Figure 1.6, half immersion is used after the initial slot operation. At the corner portion, the second loop tool path cuts along an arc whose radius is equal to the tool diameter, and the third loop tool path cuts along another arc whose radius is 1.5 times the tool diameter. As shown in Figure 1.6, the cutting angles are 90°, 104.47°, and 99.59° at points 1, 2, and 3 separately. When an outside corner is involved, the cutting angle is greater while cutting along the arcs than along the lines, but not by much.

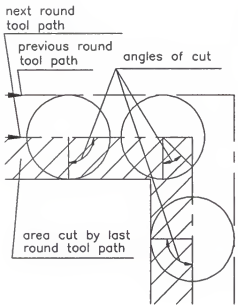


Figure 1.5 Spiral-in tool path at an inside corner

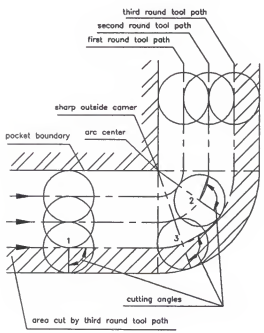


Figure 1.6 Spiral-in tool path at an outside corner

However, as the tool cuts deeper inside, the curvature of the arc will decrease and so will the increment of the cutting angle.

The third advantage of the dynamic spiral-in routine is: the tool path generated by the routine is a true spiral while the tool paths generated by the commercially available spiral-in pocketing routines are not. They contain several loops and linkages to join them. Figure 1.7 shows the difference between dynamic and conventional spiral-in routines. Notice that the cutting angle may increase when the tool is moving through the linkages on the conventional tool path. Additionally, the true spiral is more efficient in terms of machining distance (time).

Kshonze [1] and Tlusty [2] have compared the spiral in, dynamic spiral in, dynamic linear, and dynamic zig-zag pocketing routines. They found out that the dynamic spiral-in routine is by far the most efficient in terms of time required to complete a 4.5 x 4.5 x 1.125 inch square pocket. Table 1.1 shows the results of their comparison.

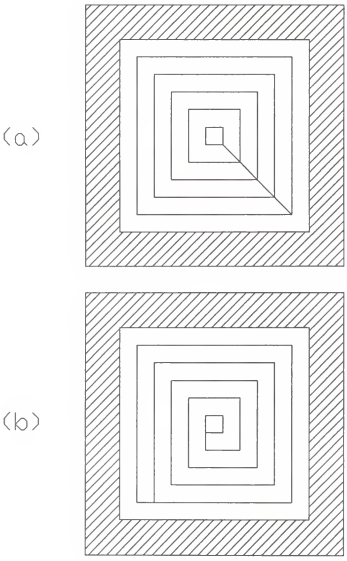


Figure 1.7 Comparison of (a) conventional and (b) true spiral-in routines

Table 1.1 Comparison of different kinds of pocketing routines for a 4.5x4.5x1.125 inch pocket

Software Package	Type of Pocket	Cutting Time (Min)	Air Travel (Min)	MRR (cm**3 /min)	# of Holes
Proposed New Routines	Spiral in	5.07	0.0	73.58	1
	Dynamic spiral in	3.62	0.0	103.07	1
	Dynamic linear without finish	4.83	0.44	70.79	1
	Dynamic linear with finish	5.65	0.08	60.96	1
	Dynamic zigzag without finish	4.20	0.08	87.18	1
	dynamic zigzag with finish	5.02	0.08	73.25	1

CHAPTER 2

OPTIMIZATION PROCESS

As mentioned in Section 1.3.1, the critical depths of cut for each tool were calculated for various cutting situations through a computer simulation program developed in the Machine Tool Laboratory (MTL). This information for many useful tools forms a database. Figure 2.1 represents the results of a simulation for a 4-flute high speed steel (HSS) end mill cutter with spindle speed at 3800 rpm and feed rate at 1900 mm/min on the White-Sundstrand milling machine at MTL. The dimension of the end mill is 25mm by 100mm. The X axis represents the radial immersion in terms of a fraction of the cutter diameter. The Y axis represents the axial immersion in millimeters. Several different cutting scenarios were created by varying the radial immersion from one fifth to full in four feed directions, +X, +X+Y, +Y, and +X-Y, for both up and down milling. Each curve splits the whole region into two sub-areas. The upper one is the unstable zone, while the lower one is the stable zone. From the figure, it is clear that the axial and radial depths of cut are strongly affected by the direction of cut as well as by the mode of milling (up or down milling). Table 2.1 lists all the data shown on Figure 2.1.

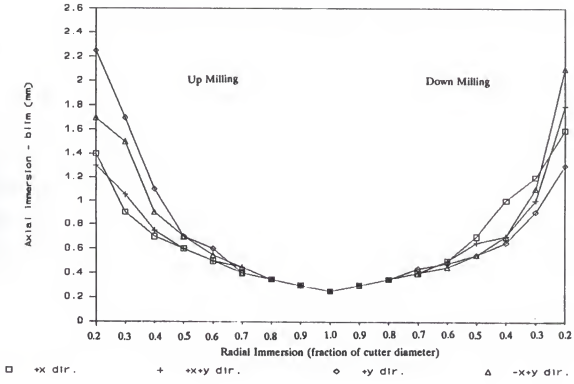


Figure 2.1 Computed permissible depths of cut for a 25mm by 100mm 4-flute HSS end mill cutter on the White-Sundstrand milling michine

Table 2.1 Permissible Depths of Cut for a 25mm by 100mm 4-Flute HSS End Mill Cutter on White-Sundstrand Machine

			Axial Immersion for Four Cutting Dir. (mm)			
			+X	+X+Y	+Y	-X+Y
Radial Immersion	Up Milling	0.2	1.4	1.3	2.25	1.7
		0.3	0.9	1.05	1.7	1.5
		0.4	0.7	0.75	1.1	0.9
		0.5	0.6	0.6	0.7	0.7
		0.6	0.5	0.5	0.6	0.55
		0.7	0.4	0.45	0.4	0.45
		0.8	0.35	0.35	0.35	0.35
		0.9	0.3	0.3	0.3	0.3
		1.0	0.25	0.25	0.25	0.25
	Down Milling	1.0	0.25	0.25	0.25	0.25
		0.9	0.3	0.3	0.3	0.3
		0.8	0.35	0.35	0.35	0.35
		0.7	0.4	0.4	0.44	0.4
		0.6	0.5	0.5	0.48	0.45
		0.5	0.7	0.65	0.55	0.55
		0.4	1.0	0.7	0.65	0.7
		0.3	1.2	1.0	0.9	1.1
		0.2	1.6	1.8	1.3	2.1

To demonstrate the use of Figure 2.1, three typical examples are presented in Table 2.2, in which the cutting directions and the axial immersions are given. In all of the examples, linear interpolation has been used. In case 1, the cutting direction is 0°, which is equivalent to +X direction, and the axial immersion is 0.65mm. The two intersections of +X curve and the horizontal line at an axial immersion equal to 0.65mm are in 0.45 in the up milling region and at 0.525 in the down milling region. These two intersections are the optimum radial immersions. In the second case, the cutting direction is 90° or +Y, and the axial immersion is 0.2mm. No intersection exists between the +Y curve and the horizontal line of axial immersion equal to 0.2mm. The situation means that there is no restriction in choosing the radial immersion. Hence the optimum radial immersion is full immersion. In case 3, the cutting direction of 22.5° is between 0° (+X) and 45° (+X+Y). Linear interpolation is used to obtain the curve for this specific cutting direction from the +X and +X+Y curves. From the new curve, the optimum radial immersions to match an axial immersion of 0.8mm are obtained by the method used in cases 1, and 2.

The objective of the optimization is to find the best combination of axial and radial immersions in machining a specific pocket to obtain the maximum metal removal rate (MRR). Two assumptions are made:

Table 2.2 Examples of finding optimum radial immersions from the given cutting directions and axial immersions

Cutting Direction (Degree)	Axial Immersion (mm)	Opti. Radial Immersion (% of Tool Dia.)	
		Up milling	Down Milling
0	0.65	0.45	0.525
90	0.2	1.0	1.0
22.5	0.8	0.37	0.429

1) The pocket will be machined in several layers with the same thickness between two adjacent layers;

2) At each layer, the axial depth of cut remains constant.

The method of optimization is explained below.

when the number of layers, say N_L, is determined, the axial depth of cut, b, will be

$$b = D_{pocket} / N_L \qquad (2.1)$$

where D_{pocket} is the depth of the pocket. As shown in Table 2.2, if the axial depth of cut, b, is given, then the corresponding optimum radial immersions for any cutting directions can be obtained. Since the machining time required to finish a layer, T_L, is proportional to

$$T_L \propto \frac{1}{(a_1 p_1 + a_2 p_2 + \ldots + a_n p_n)} \qquad (2.2)$$

where n is the number of pocket edges,

a_1, a_2, ..., a_n are the corresponding optimum radial immersions of the pocket edges for the

specific axial immersion, b,

p_1, p_2, ..., p_n are the corresponding percentages of the edge lengths in total pocket boundary length. They are constants.

The machining time for the whole pocket, T_p is

$$T_p = T_L N_L \propto \frac{N_L}{(a_1 p_1 + a_2 p_2 + \ldots + a_n p_n)} \tag{2.3}$$

Since MRR is proportional to $1/T_p$, it is

$$MRR \propto \frac{1}{T_p} \propto \frac{(a_1 p_1 + a_2 p_2 + \ldots + a_n p_n)}{N_L} \tag{2.4}$$

The optimization process will search through a range of N_L. From each N_L, the axial depth of cut, b, and the corresponding optimum radial depths of cut for this b are determined. Thus, the value of Equation 2.4 can be obtained. The N_L, that produces the largest value from Equation 2.4, will be the optimum number of layers. The corresponding b, and a_1 to a_n are the optimum axial and radial depths of cut.

For example, if the maximum and minimum allowable radial immersions are 0.7 and 0.2 separately, from Table 2.1, the range of b will thus be determined as between 0.4 mm and 1.3 mm for both down and up milling cases. For using values of b greater than 1.3 mm, the tool may chatter in some cutting directions. For using values of b smaller than 0.4 mm, the machining may not achieve the desired efficiency. If the depth of the pocket, D_{pocket}, is 20 mm, then the number of

layers, N_L, needed to machine the pocket is in the range of 16 to 50 (20mm/1.3mm and 20mm/0.4mm). For each N_L, the corresponding axial depth of cut, b, is determined by Equation 2.1, and then the optimum a_1 to a_n for this b can be obtained by the method shown in Table 2.2. After that, the value of Equation 2.4 is calculated. The optimum N_L and its corresponding b and a_1 to a_n will produce the largest value.

In spiral pocketing, the tool paths are determined from pocket boundaries by successively shrinking the periphery of the pocket segment-wise by the permissible radial depth of cut for that segment. The result of each such successive offset is a sub-loop of the previous loop. Although the ratio of an edge in total length of a certain loop may not be the same as the ratio of the corresponding edge in total length of its sub-loop, the difference will not be much. Since in a pocketing operation, a few outer loops would cut most of the material, and thus they count the most, it is reasonable to assume that the ratios are the same in these few loops. Therefore, although the optimization process is a simplified method, and may not obtain the true optimum solution, it certainly would achieve one of the better results.

CHAPTER 3

PROGRAM STRUCTURE

While developing a program with a vast amount of entangled data, a method has to be developed to keep track of all data so that nothing is neglected. That is why a data structure is necessary. Standard data structures include stack, array, linked list, tree, etc. However, frequently in a program these types are used jointly. For example, the node of a linked list might be an array, or the node of a tree structure might be a linked list. A distinct data structure should be developed for each program to serve its own purpose. All the example codes in this chapter are written in C programming language or pseudo codes which are sometimes easier to understand.

3.1 Data Representation of a Segment

Each segment, a line or an arc, can be represented by a set of geometric data.

An array storing the coordinates of the starting and the ending points are enough to represent a line. In this case an array of four elements is adequate. The first and the second element of the array will keep the X and Y coordinates of the

starting point, while the third and the fourth elements store the coordinates of the ending point.

For an arc, an array of five elements is required, where the first and the second elements are used for holding the arc center coordinate, the third element representing the arc radius, and the fourth and the fifth elements storing the starting and the ending angle with respect to the x axis.

Since it is simpler to use the same scheme to represent all kinds of segments, an array of five is used to hold the geometric data for both a line and an arc, although the last element of the array is idle when representing a line.

Another variable, called TYPE, is required. Its value is used to distinguish the type of segment. If the value of the variable TYPE is 2, then the segment is a line. Variable TYPE with a value of 5 represents a counterclockwise (CCW) arc, and 4 represents a clockwise (CW) arc.

A line segment with (0.0,1.0) and (10.0,11.0) as the starting and ending coordinates is represented as follows:

```
type = 2,                (a line segment)
dt[0] = 0.0,
dt[1] = 1.0,
dt[2] = 10.0,
dt[3] = 11.0.
```

An clockwise arc with center coordinate (1.0 1.0), radius 2.0, starting angle π, and ending angle $\pi/2$, is represented as follows:

```
type = 4,                (a CW arc)
dt[0] = 1.0,
dt[1] = 1.0,
dt[2] = 2.0,
dt[3] = π,
dt[4] = π/2.
```

3.2 Linked List Representation

A linked list is a sequence of zero or more nodes, together with a set of operations. Each node contains two fields, an ITEM field that holds information and a NEXT field that holds a pointer to the next node in the list. The fact that each node contains a specification of where the next node is, makes it a linked list. The name of the linked-list is simply an indication of where the first node may be found; all other nodes must be obtained by following the chain of links [11].

The conventional way to portray a linked list is with a box-and-arrow diagram. Each node is shown as a box with an ITEM part on the left and a NEXT part on the right. An arrow leads from the NEXT part to the next node and the last node in the list is shown as a box in which a diagonal line is drawn through the NEXT part to indicate that there isn't any next node. Note that the arrow runs in only one direction. This is a singly-linked list [11].

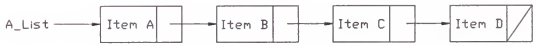


Figure 3.1 A symbolic sketch of singly-linked list

In many applications it is convenient to be able to move through a list in either direction. The answer is to put two link fields in each node, one pointing to the successor and one to the predecessor of the node. The representation is called a doubly-linked list. Each node of a doubly-linked list contains three fields. Besides the ITEM and NEXT, a PREVIOUS field is added which holds a pointer to the previous node in the list. This convenience is not free. Besides the memory space required for the extra links, more links need to be rearranged and extra complexity is added while manipulating the doubly-linked list.

Another variation is that the link in the last node, instead of containing an indication that it is the last node, can point back to the head of the list. This is called a circularly-linked list.

Since a closed loop consists of many segments with one trailing the other, it is ideal to use a circularly-doubly-linked list to represent a closed loop.

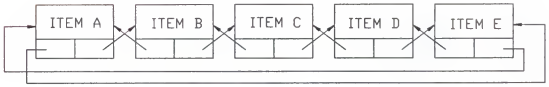


Figure 3.2 A symbolic sketch of a circularly-doubly-linked list

In programming languages, there are two ways to implement a linked list. The older one is called array implementation, and the newer and better one, is called pointer implementation since it relaxes the restriction of prespecifying the sizes of arrays. The Fortran language only allows the former one.

In C programming language, the box of the circularly-doubly-linked list is a structure variable which is defined by a customized structure template. The syntax of defining a structure template, called "segment", is shown as follows:

```
typedef struct a_segment
{ int     type;
  double  dt[5];
  struct  a_segment  *next;
  struct  a_segment  *prev;
  struct  a_segment  *origin;
} segment
```

This structure template contains an integer variable "type", an array "dt" of five double-precision real numbers, and three pointer variables named "next", "prev", and "origin" separately. Among the variables, "type" and "dt" are the "ITEM" part of the linked list, their purpose being already stated in Section 3.1.

The statement that causes a structure variable "a_seg" to be created according to the above structure template is

```
segment  *a_seg;
```

Also the following statement is needed that will allocate a memory block and will assign the address of the memory block to the structure variable "a_seg".

```
a_seg = (segment *) calloc (1, sizeof(segment));
```

After that, "a_seg" is a structure variable which holds the address of the newly allocated memory block. In other words, the structure variable "a_seg" is a pointer pointing to the memory block.

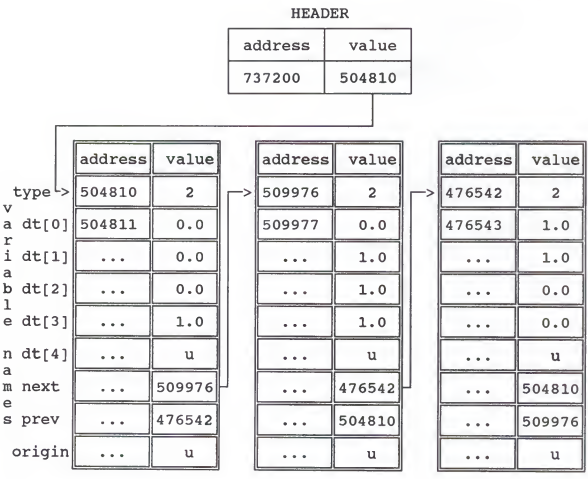
A memory block occupies many continuous memory locations, 24 continuous memory locations at 48 bytes in this case, and its address is also the address of the first memory location. The pointer variables next, prev, and origin will hold the addresses of the other memory blocks.

The access of a structure member is through the structure member operator "->". For example, to assign a value of 2 to the integer variable "type" of the structure variable "a_seg", the C language syntax is

```
a_seg->type = 2;
```

Assume that the three vertices of a clockwise triangle are denoted as (0,0), (0,1), and (1,1) separately, and the line segment from (0,0) to (0,1) is the starting segment of the triangle. After proper assignment of this data to the appropriate memory locations in three memory blocks, in Table 3.1, three charts in double lines show how the memory blocks hold data. The variables in each memory block include those needed to represent a line segment which are variables "dt" and "type" (refer to Section 3.1), and variables "next", "prev", and "origin". Notice how variables "next" and "prev" save the addresses of the next and previous memory blocks. For example, the value in variable "next" of the first memory block is 509976 which is also the address of the second memory block. The value in variable "prev" of the first memory block is 476542 which is also the address of the last memory block. By following the same rule, it is easy to find out how the memory blocks interact with each other. In Table 3.1, the single lines show some examples of interaction between addresses and values. Each time when the memory allocation statement is invoked, the computer will allocate the available memory blocks. Hence the addresses are pseudo numbers in Table 3.1. Note that the name of the structure variable, which points to the first memory block, is the header whose value is the first memory address of the block. From this particular address, all data are retrievable.

Table 3.1 An example of data storage in a circularly-doubly-linked list



* u : unassigned.

3.3 Tree Representation of a Pocket and its Spiral Tool Paths

A pocket profile and its tool paths are a number of closed loops constituted by several segments (line segments or arcs). The outer loop is called the parent and it inner loops are the children. A parent can have several children, but any child can only have one parent. All loops can have no intersection between each other.

Figure 3.3a shows a topological representation of a pocket and its spiral tool paths. In the figure the outermost loop, loop A, symbolizes the pocket profile. The other loops represent the tool paths. Loop A has one child, which is loop B, loop B has three children, called loop C, loop D, and loop E respectively, and etc. It is immediately recognizable that a general tree representation will perfectly match Figure 3.3a. Figure 3.3b is the equivalent general tree representation of Figure 3.3a. In Figure 3.3b, node A has one child, called node B, node B has three children which are node C, node D, and node E respectively, and etc.

To explain the "general tree" better, the idea of a "binary tree" has to be mentioned first. The definition of a binary tree is: a binary tree is a set of nodes that is either empty, or consists of a root and two disjoint binary trees called the left sub-tree and the right sub-tree. A node contains information, the nature of which depends on the application. [11]

The difference between a "general tree" and "binary tree" is that the general tree relaxes the requirement that each tree node has exactly two children. Instead, the general tree can simply have zero or more children.

Binary trees can be easily used to represent general trees. The basic idea of representation is to order the children of a general tree node, say, from left to right. Now in the binary tree representation, each has two link fields, which are called FIRST and NEXT, rather than LEFT and RIGHT. The FIRST field of a node points to the first (leftmost) son of the general tree node. And the NEXT field points to the next sibling node. The correspondence between these two tree representations is illustrated by Figure 3.3b and 3.3c. Figure 3.3c is the equivalent binary tree for Figure 3.3b. Here in Figure 3.3c, the first child of node A is node B, and the first child of node B is node C. Both node A and node B have no siblings. But node C has two branches, one pointing to its first child, node F, and the other to its next sibling, node D. Node D, which is the same as node C, is also a child of node B, and etc.

There are several advantages in using the binary tree data structure in representing tool paths. For example, the tree traversal by using the recursive method, which will be mentioned in Section 3.6, can be used to create the tool path for any exotic pocket with as many islands as desired.

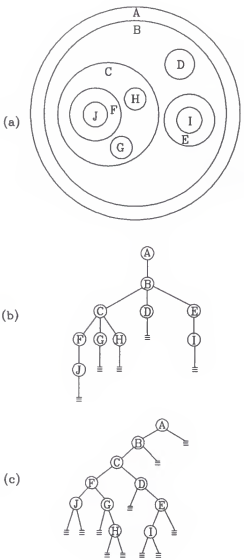


Figure 3.3 Representing tool path as a tree
(a) topological representation of a pocket and its tool paths
(b) the equivalent general tree
(c) the equivalent binary tree

3.4 Program Implementation of a Binary Tree

Like a linked list, there are two ways to implement the binary tree representation, one called the array implementation, the other known as a pointer variable implementation. Here the latter one is chosen.

First a particular structure template named "node" for the tree node is defined as follows:

```
typedef struct nd
{ int          nd_type;
  int          nd_index;
  struct  nd   *first;
  struct  nd   *next;
  struct  nd   *parent;
  struct  seg  *start;
} node;
```

If a structure variable "a_node" is designated according to template "node", the statement is

```
node  *a_node;
```

The next statement will allocate a memory block and then will assign its address to the variable "a_node". The syntax is

```
a_node = (node *) calloc (1, sizeof(node));
```

After that, "a_node" is a pointer pointing to the newly allocated memory block.

The pointer variable "first" of the structure holds the address of a structure representing its first child, and "next" holds the address of another structure representing its

next sibling. These two pointer linkages of each node make the connection among general tree nodes.

The other pointer variables include "start" and "parent". The pointer variable "parent" of the structure points to the memory block representing the parent node. This variable is useful in finding the passages between loops.

The pointer variable "start" of the structure is the header of a circularly-doubly-linked list (refer to Section 3.2). In other words, "start" holds the address of the first memory block of the circularly-doubly-linked list. The linked list holds all the data representing a closed loop. From pointer "start", any data at any segment on the closed loop is retrievable. For example, if variable "an_integer" is assigned the value of variable "type" at the second segment of the closed loop, the following statement will do the work:

```
an_integer = a_node->start->next->type;
```

Here, the second segment is interpreted as the one next to the starting segment. The value of variable "type" of the third segment will be obtained by the following statement:

```
a_node->start->next->next->type;
```

The value of "type" of the last segment will be obtained by:

```
a_node->start->prev->type;
```

Here, the last segment is the one previous to the starting segment. Usually a do-while loop in the computer program is used to reach all segments in a loop.

Two integer variables nd_type and nd_index of the template will indicate the validity of the tree node.

The relationship of the tree nodes and the doubly-linked list is shown by Figure 3.4. Figure 3.4a illustrates the tool paths for a pocket and Figure 3.4b depicts the matching tree and linked lists. The figure shows the following relations:

```
Loop A -> first = Loop B;
Loop A -> next = NULL;
Loop A -> start = Segment 1;
Loop B -> first = Loop C;
Loop B -> next = NULL;
Loop B -> start = Segment 9;
Loop C -> first = NULL;
Loop C -> next = Loop D;
Loop C -> start = Segment 19;
Loop D -> first = NULL;
Loop D -> next = NULL;
Loop D -> start = Segment 24;
Segment 1 -> next = Segment 2;
Segment 1 -> prev = Segment 8;
Segment 2 -> next = Segment 3;
.... etc.
```

3.5 Island Representation

The islands are a string of independent closed loops within the pocket boundary and without any intersection among

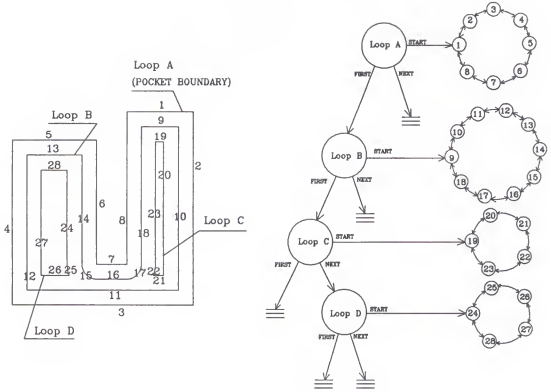


Figure 3.4 A example of tool paths for a pocket and its matching tree and linked lists

them. The islands are represented by another linked list. Each node of the linked list represents an island profile.

In the programming language, the node of the linked list is a pointer variable which points to a memory block or in other words holds the value of the address of the memory block. The memory block will contain two pointer variables, "next" and "start". Pointer variable "next" points to the next island. Pointer variable "start" points to the starting segment of the island profile. All segments of an island profile will form a circularly-doubly-linked list.

Instead of defining another structure template for island representation, the same structure template, used to represent the tree nodes, is used in representing the nodes of the linked list for islands. The trick is that all other pointer variables, which are useless for island representation, are given a value of NULL.

In the beginning, using the same structure template in representing both islands and tree nodes may waste a few unnecessary memory spaces. But overall the program will be shorter and simpler because there are fewer cases to distinguish, and thus the executable file is smaller and computational time is saved.

Figure 3.5a demonstrates a pocket with four islands, and Figure 3.5b shows the representation of those island profiles. Note that the left side of Figure 3.5b is a linked list. Island 1 points to Island 2, Island 2 points to Island 3, and

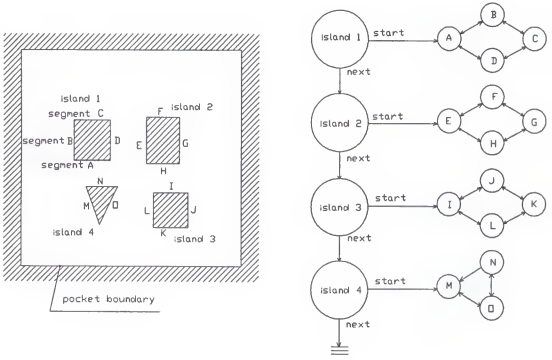


Figure 3.5 An example of pocket containing islands and the linked list representation of the islands

so on. The last island, Island 4 will point to NULL. The right side of Figure 3.5b contains four circularly-doubly-linked lists. The relationship of those nodes can be written by the following expressions:

Island 1 -> next = Island 2,

Island 2 -> next = island 3,

.....

Island 4 -> next = NULL.

Also,

Island 1 -> start = Segment A,

Island 2 -> start = Segment E,

.....

Island 4 -> start = Segment M.

3.6 Tree Traversal

The tree traversal means "visiting" all the nodes of a tree. Here visiting a node may mean cutting along a loop in the pocket. The traversal of a binary tree is approached through the recursive definition. There are three generally used tree traversals:

(a) preorder traversal:

1. Visit the root.
2. Traverse the left sub-tree.
3. Traverse the right sub-tree.

(b) Inorder traversal:

1. Traverse the left sub-tree.

2. Visit the root.

3. Traverse the right sub-tree.

(c) Postorder traversal:

1. Traverse the left sub-tree.

2. Traverse the right sub-tree.

3. Visit the root.

But in the general tree representation, first and next are used instead of left and right. Consider the tree of Figure 3.3b, The three traversals are:

(a) Preorder traversal: A B C F J G H D E I.

(b) Inorder traversal: J F G H C I E D B A.

(c) Postorder traversal: J H G F I E D C B A.

While the representative tree is completely constructed, moving through a tree node may mean cutting around a closed loop. In this case, the preorder traversal can be treated as the cutter moving from outside in, while the inorder traversal is the cutter moving from inside out. The postorder traversal is meaningless in this tree representation.

Since the preferred tool path is from outside to inside, only the programming implementation of preorder traversal is mentioned here. The first step is to create a procedure named "cut_around_loop", which will direct the cutter moving along a closed loop. After that the procedure, named "cut_into", will complete the cutter movement as desired. Procedure "cut_into" is listed as follows:

cut_into (root_node)

```
node *root_node;
{ if (root_node != NULL)     /* root_node not equal to NULL */
  { cut_around_loop (root_node);
    cut_into (root_node->first);
                        /* procedure cut_into calls itself */
    cut_into (root_node->next);
                        /* procedure cut_into calls itself */
  } }
```

Notice that a recursive method is used here. Recursion, as a programming technique, comes into play any time a procedure calls itself. Recursion makes a program easier to write and to understand than its nonrecursive counterparts.

The other usage of preorder traversal is in constructing a tree map. First a subroutine, called "find_children", is needed. The purposes of the subroutine are to obtain all the sub-loops from a larger loop and to construct pointer linkages between this loop and the resulting sub-loops. Preorder traversal technique provides a way to reach every node, and then the subroutine "find_children" can be used to find all of its offsprings. Preorder traversal will again treat those offsprings as larger loops and "find_children" would find their offsprings. By continuing this process the tree map will be fully constructed. The following subroutine, called "construct_tree", is designed to perform the process mentioned above.

```
construct_tree (root_node)
node  *root_node;
{ if (root_node != NULL)    /* root_node not equal to NULL */
  { find_children (root_node);
    construct_tree (root_node->first);
    construct_tree (root_node->next);
  } }
```

Again, a recursion technique is used here. When the above subroutine is invoked, Figure 3.6 shows the sequence of constructing a tree from the outermost loop to obtain a tree map of Figure 3.3c. The above two programs have been simplified to explain the concept of using preorder traversal, although the real ones have more details to consider. Another place that the preorder traversal method is used is in finding the passages between closed loops described in Section 5.5.

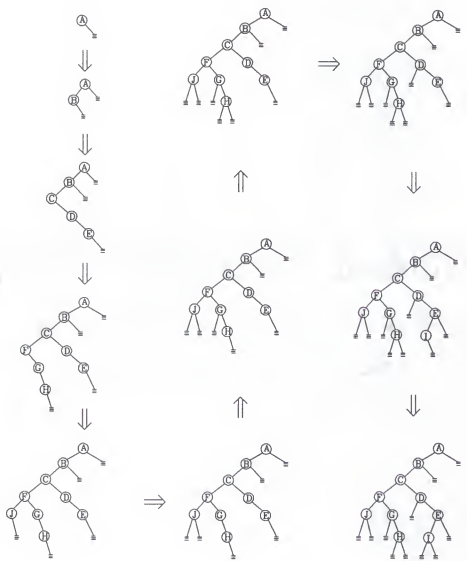


Figure 3.6 The tree construction sequence by subroutine "construct_tree"

CHAPTER 4

SOME SUPPORTING SUBROUTINES

4.1 Classifying Inside-Outside Relation of a Point and a Closed Loop

The winding number [12] reveals the relationship between a closed loop and an arbitrary point. Three cases are possible as shown in Figure 4.1. The winding number equals 1 in the first case. It happens when a point p_0 is inside the closed loop and the loop is CCW. The second case is one in which the point p_0 is inside the loop and the loop is CW. Under this circumstance, the winding number is equal to -1. The third case is when p_0 is outside the loop. Then no matter what the direction of the loop, the winding number equals 0. The winding number of a closed loop is denoted as follows.

$$W = \sum_{i=1}^{n} W_i, \tag{4.1}$$

where n is the number of segments which construct the closed loop. And

$$W_i = \frac{1}{2\pi}\int \frac{(x_i(t)-P_x)\ y_i'(t) - (y_i(t)-p_y)\ x_i'(t)}{(x_i(t)-p_x)^2 + (y_i(t)-p_y)^2}\,dt \tag{4.2}$$

where (p_x,p_y) is the coordinate of point p_0, $(x_i(t),y_i(t))$

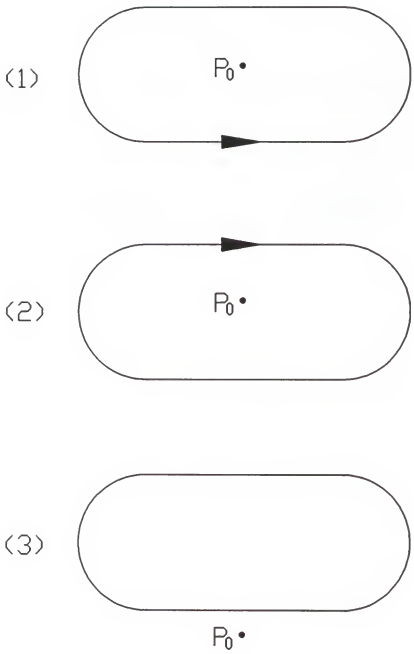


Figure 4.1 Three cases of winding number
(a) winding number = +1
(b) winding number = -1
(c) winding number = 0

represents the coordinate of any point along the i'th segment of the loop, and $x_i'(t)$ and $y_i'(t)$ are the first derivatives of $x_i(t)$ and $y_i(t)$ respectively with respect to t.

If the i'th segment is a line segment which has (a_1,b_1) and (a_2,b_2) as the starting and ending point, then it is represented as:

$$x_i(t) = a_1 + t\,(a_2-a_1)\ ,$$

$$y_i(t) = b_1 + t\,(b_2-b_1), \qquad for\ 0 \le t \le 1 \tag{4.3}$$

If the segment is an arc, it is represented as:

$$x_i = x_c + r\cos(t),$$

$$y_i(t) = y_c + r\sin(t), \quad \begin{matrix}\theta_1 \le t \le \theta_2 & for\ CCW\ \text{arc} \\ \theta_1 \ge t \ge \theta_2 & for\ CW\ \text{arc}\end{matrix} \tag{4.4}$$

where (x_c,y_c) is the arc center coordinate, r is the arc radius, θ_1 is the starting angle, and θ_2 is the ending angle. For example, if the arc is counterclockwise spanning from 0 to 0.5π, then

$$\theta_1 = 0, \quad \theta_2 = 0.5\ \pi,$$

or if it is a clockwise arc spanning from 0.5π to 1.5π, then

$$\theta_1 = (0.5 + 2)\ \pi = 2.5\ \pi, \quad \theta_2 = 1.5\ \pi.$$

The exact solution of W_i for the i'th segment can be obtained by integral if the segment is an arc or a line segment. But if other kinds of curves are involved, numerical integration may have to be applied. The computational results of the winding number W_i are shown as follows:

(A) For line segment

Let constants c_1, c_2, and c_3 be

$$c_1 = (a_1 - p_x)(b_2 - b_1) - (b_1 - p_y)(a_2 - a_1),$$

$$c_2 = (a_2 - a_1)^2 + (b_2 - b_1)^2,$$

$$c_3 = 2(b_2 - b_1)(b_1 - p_y) + 2(a_2 - a_1)(a_1 - p_x).$$

The winding number W_i will be

$$W_i = 0, \qquad for\ c_1 = 0 \tag{4.5}$$

$$= \frac{c_1}{2\pi}\left[\tan^{-1}\frac{2c_2 + c_3}{|2c_1|} - \tan^{-1}\frac{c_3}{|2c_1|}\right], \qquad for\ c_1 \neq 0 \tag{4.6}$$

(B) For arc

Let constants

$c_1 = x_c - p_x$,

$c_2 = y_c - p_y$,

$c_3 = r^2 + c_1^2 + c_2^2$,

$c_4 = 2\ r\ (c_1^2 + c_2^2)^{0.5}$,

$\theta_3 = \tan^{-1}(c_1/c_2)$ when $c_2 > 0$, or

$\theta_3 = \tan^{-1}(c_1/c_2) + \pi$ when $c_2 < 0$, or

$\theta_3 = 0.5\pi$ when $c_2 = 0$ and $c_1 > 0$, or

$\theta_3 = -0.5\pi$ when $c_2 = 0$ and $c_1 < 0$.

The winding number W_i will be

$$W_i = \frac{(\theta_2 - \theta_1)}{4\pi} \qquad if\ (r^2 - c_1^2 - c_2^2) = 0 \tag{4.7}$$

or

$$W_i = \frac{(\theta_2 - \theta_1)}{2\pi} \qquad if\ c_1 = c_2 = 0 \tag{4.8}$$

Otherwise

$$W_i = \frac{\theta_2-\theta_1}{4\pi} + \frac{r^2-c_1^2-c_2^2}{|r^2-c_1^2-c_2^2|} \tan^{-1}\frac{c_3 \tan(0.5m) + c_4}{r^2-c_1^2-c_2^2} \Bigg|_{m=\theta_1+\theta_3}^{m=\theta_2+\theta_3} \tag{4.9}$$

Notice that, because of the limitation of the function $\tan^{-1}$, the range of m should always be between $-\pi$ and π. Thus, in the case of m ranging from 0 to 1.5π, it is necessary to change its range to from 0 to π plus from $-\pi$ to -0.5π. Additionally, it has to be kept in mind that the value of the $\tan^{-1}$ part of the Equation 4.9

$$\tan^{-1}\frac{c_3 \tan(0.5m) + c_4}{r^2 - c_1^2 - c_2^2} = \frac{\pi}{2} \qquad if\ m = \pi,$$

$$= -\frac{\pi}{2} \qquad if\ m = -\pi$$

Notice that since the exact solution instead of the numerical approximation is used, the result is very accurate even if the chosen point is very close to the loop. The error usually is between ± 0.001.

The winding number is used to classify the direction of a closed loop as well as to distinguish the inside-outside relationship between a certain point and a loop.

The reason to check the orientation of a loop is that the validity of a loop is determined by its direction. While obtaining the sub-loop from a previous loop, the geometry may cause a crossing between offsets. In such a case, one outer loop generates two or more sub-loops. If a pocket consists of

islands, they are included by successively expanding their boundaries segmentwise. Again intersections may cause multiple sub-loops. In both cases, only those sub-loops with the same orientations as its larger loop are valid. Those with different orientation are eliminated and it is not necessary to find their sub-loops in the next path generation process.

Since a point is needed while calculating the winding number, in order to find out the orientation of a loop, an appropriate point has to be chosen. If the calculated winding number can not be used to determine the orientation, it is necessary to try another point. The procedure of classifying the orientation is:

(1) Find the mid-point of the first segment of the loop boundary.

(2) Find the point which is at a distance of +0.05 times the segment length from the mid-point and on the line perpendicular to the segment at the mid-point.

(3) Find the winding number. If it is between +1.001 and +0.999, the loop is CCW and the program is terminated. If it is between -1.001 and -0.999, the loop is CW and the program is terminated. If neither is the case, the procedure needs to be continued.

(4) Repeat step 2 except the distance is -0.05 times the segment length, and then perform step 3 for the new point.

(5) Find the mid-point of the next segment until all the

segments are reached, and then repeat step 2 to step 4.
(6) If the orientation still can not be determined, it can be claimed that the loop is too small to be a valid loop.

The following case is another example of using the winding number. If a sub-loop, obtained from continuously shrinking pocket boundaries, is inside the expanded loop of an island boundary, then this loop is invalid. The situation of a closed loop inside another closed loop should satisfy the condition that each segment in the first loop is inside the second loop. The condition of a segment being inside a closed loop can be checked by the following two conditions:
(1) No intersection exists between the segment and the loop, or if intersections did exist, they are at the end points or at the tangent.
(2) An appropriate point on the segment is inside the loop. The winding number calculation comes in handy to verify the second condition.

4.2 Intersection of Two Segments

Three kinds of intersection are expected which are intersection of two line segments, of a line segment and an arc, and of two arcs.

The procedure to find the intersection between two line segments involves the following three steps:
(1) Find the unlimited lines which contain the line segments.
(2) Find the intersection between these two unlimited lines.

(3) Check the found point to verify whether it belongs to both line segments.

The procedure to find the intersection between a line segment and an arc involves the following steps:

(1) Find the unlimited line which contains the line segment, and the circle which encloses the arc.

(2) Find the intersection(s) between the unlimited line and the circle.

(3) Check the found point(s) to verify whether it belongs to both the line segment and the arc.

The procedure to find the intersection between two arcs is arranged by the following steps:

(1) Find the intersection(s) between two circles which have the same center coordinates and radii as the two arcs.

(2) Verify whether the found point(s) is on both arcs.

Several supporting subroutines are needed for the above three procedures. Their names are: "line_dir", "point_on_line_segment", "point_on_arc", "line_2_points", "intersection_2_lines", "intersection_line_circle", and "intersection_2_circles".

The purpose of subroutine "line_dir" is to obtain the direction angle of a line segment with (x_1,y_1) and (x_2,y_2) as the coordinates of the starting and ending points.

Subroutine "point_on_line_segment" has the arguments x_1, y_1, x_2, y_2, x_3, and y_3. The line segment is denoted by two end points with coordinates (x_2, y_2) and (x_3, y_3). If point $(x_1,$

y_1) is on the line segment, the subroutine returns TRUE, and otherwise FALSE. Note that even if the point is not on the line segment, but with a distance of less than 0.00005 inch, then the subroutine will return TRUE, since in most NC milling machines the accuracy is only up to 10^{-4} inch. The clearance of 0.00005 inch also applies to the subroutine "point_on_arc".

Subroutine "line_2_points" will return the coefficients a, b, and c of the line equation

$$ax + by + c = 0.$$

The input arguments are the coordinates of two end points, (x_1, y_1), and (x_2, y_2).

Subroutine "intersection_2_lines" will return the coordinate of the intersection point. The input arguments are a_1, b_1, c_1, a_2, b_2, and c_2 which represent two unlimited lines

$$a_1x + b_1y + c_1 = 0,$$

and

$$a_2x + b_2y + c_2 = 0.$$

Subroutine "intersection_line_circle" will return the number of intersecting points (0, 1, or 2) and their coordinates if the intersection exists. The input arguments are a, b, c, x, y, and r. Arguments a, b, and c are the coefficients of a line equation representing an unlimited line, and (x, y) and r represent the center coordinate and radius of a circle.

The subroutine "intersection_2_circles" will return the number of intersecting points (zero, one, or two) and their

coordinates if the intersection exists. The input arguments are the center coordinates and radii of these two circles.

The algorithms of the above two subroutines are built by the geometric relation between segments rather than by solving the roots of two equations. The method of finding the intersection of a line and a circle is listed as follows:

Given: the coefficients a, b, and c of equation $ax+by+c=0$ which represents Line A, the center coordinate (x_0, y_0), and the radius r_0 of the circle. All notations are defined in Fig. 4.2.

Solution:

(1) Find d, the distance between (x_0, y_0) and Line A,

$$d = \frac{| ax_0 + by_0 + c |}{(a^2 + b^2)^{0.5}}$$

(2) If $d > r_0 + 0.00005$, no intersection exists.
Else if $r_0 - 0.00005 \leq d \leq r_0 + 0.00005$, n = 1. Here n is the number of intersection point.
Else n = 2.

(3) Assume that Line B is the line which passes through (x_0, y_0) and is perpendicular to Line A. The equation of Line B is

$$-bx + ay + (bx_0 - ay_0) = 0.$$

Find the intersection, point (x_1, y_1), of Lines A and B.

(4) If n = 1, the coordinate of the intersecting point between Line A and the circle is (x_1, y_1). Else if n=2, then

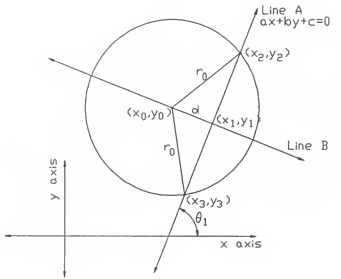


Figure 4.2 Line-circle intersection

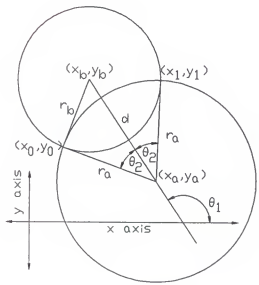


FIgure 4.3 Circle-circle intersection

(i) find θ_1, the direction angle of $ax + by + c = 0$;

(ii) the intersection points (x_2, y_2) and (x_3, y_3) are

$$x_2, x_3 = x_1 \pm (r_0^2 - d^2)^{0.5} \cos(\theta_1),$$

$$y_2, y_3 = y_1 \pm (r_0^2 - d^2)^{0.5} \sin(\theta_1),$$

The method of finding the intersection of two circles is listed as follows:

Given: (x_a, y_a) and r_a, the center coordinate and radius of Circle A, and (x_b, y_b) and r_b, the center coordinate and radius of Circle B. All notations are defined in Figure 4.3.

Solution

(1) Find the distance between two centers,

$d = ((x_b - x_a)^2 + (y_b - y_a)^2)^{0.5}$.

(2) If $d > (r_a + r_b + 0.00005)$, then $n = 0$. Here n is the number of intersection points.

Else if $(r_a + r_b - .00005) \leq d \leq (r_a + r_b + .00005)$, then $n = 1$.

Else if $d < (|r_a - r_b| - .00005)$, then $n = 0$.

Else if $(|r_a - r_b| - .00005) \leq d \leq (|r_a - r_b| + 0.00005)$, then $n = 1$.

Else $n = 2$.

(3) Find θ_1, the direction angle of the vector from (x_a, y_a) to (x_b, y_b).

(4) If $n = 1$, $\theta_2 = 0$.

Else if $n = 2$, then $\theta_2 = \cos^{-1}\left(\frac{r_a^2 + d^2 - r_b^2}{2\, r_a\, d}\right)$.

The coordinates (x_0, y_0) and (x_1, y_1) of the intersection points are

$$x_0,\ x_1 = x_a + r_a \cos(\theta_1 \pm \theta_2),$$

$$y_0,\ y_1 = y_a + r_a \sin(\theta_1 \pm \theta_2).$$

CHAPTER 5

FINDING THE TOOL PATH OF A POCKET

The tool path for a pocketing operation can be divided into two portions: One is the initial slotting pass around the pocket profile as well as around the island profiles, if islands exist. The other is the spiral machining pass from the outside in. The tool path of the slotting pass is the shrunk offset loop from the pocket boundary with the distance being expressed as a sum of the tool radius plus the desired finishing immersion. On the contrary, the expanded offset loops from island profiles are the slotting tool path for islands. Thus, the tool passes for slotting are found by using equal offset distance. For the spiral-in tool path, the optimization process, mentioned in Chapter 2, can be used to determine the radial immersions. Each machining direction will use an immersion which may be different from that of the other machining directions. These optimized immersions are the offset distances for finding the continuous shrunk offset loops from the slotting pass loop. These offset loops, found by using unequal offset distances, become the spiral-in tool passes.

5.1 Finding Offset Loops of Equal Offset Distance

The process of finding an offset loop from a closed loop is comprised of three steps as follows:

1. finding the offset of each segment on the original loop.
2. linking pair by pair the offset segments whose original segments are adjacent thus resulting in a gross offset loop.
3. splitting the gross offset loop into several sub-loops.

These steps are explained in detailed below.

5.1.1 Finding the Offset of Each Segment on the Original Loop

The offset of a line segment is a parallel line segment at an offset distance. Figure 5.1 shows the relationship of a line segment L_1 and its two offset segments, O_1 and O_2. Segment L_1 starts from (a, b) and ends at (c, d). The offset distance is D. The starting points of O_1 and O_2 are

$$\begin{matrix} O_{1,start} \\ O_{2,start} \end{matrix} = \left(\frac{\pm D(b-d)}{\sqrt{(c-a)^2+(d-b)^2}} , \frac{\pm D(c-a)}{\sqrt{(c-a)^2+(d-b)^2}} \right) \qquad (5.1)$$

The ending points of O_1 and O_2 are

$$\begin{matrix} O_{1,end} \\ O_{2,end} \end{matrix} = \left(\frac{\pm D(b-d)}{\sqrt{(c-a)^2+(d-b)^2}}+c-a , \frac{\pm D(c-a)}{\sqrt{(c-a)^2+(d-b)^2}}+d-b \right) \qquad (5.2)$$

Since, in Section 3.1, it has been defined that the coordinates of the starting and ending points are used to represent a line segment, offset segments O_1 and O_2 are determined by the above expressions.

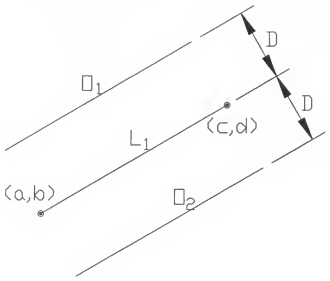


Figure 5.1 The relationship of a line segment, L_1, and its offset segments, O_1 and O_2

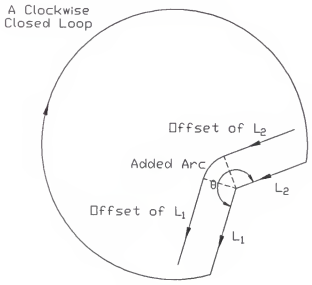


Figure 5.2 Adding extra arc to connect two offsets

The offset segment of an arc is another arc of the same orientation, either CW or CCW. As defined in Section 3.1, geometrically an arc is represented by its center point coordinates, its radius, and the starting and the ending angle with respect to the X axis, and topologically an arc is "type" 4 if CCW and "type" 5 if CW. The "type" of the offset arc will remain the same as its original arc. Geometrically, the offset arc will be represented by the same center point coordinates, starting angle, and ending angle as the original arc. However, the radius of the offset arc is an offset distance greater or smaller than the original arc. There are cases when the radius of the original arc is smaller than the desired offset distance. In this circumstance, the arc has no inner offset arc.

5.1.2 Linking the Offset Segments Whose Original Segments Are Adjacent

The linking process between two offset segments can be separated into two cases. These two cases are distinguished by the angle between the neighboring parent segments on the original closed loop, this angle being greater or lesser than 180°.

5.1.2.A Case 1: the Angle Is Greater Than 180°

As shown in Figure 5.2, since θ is greater than 180°, an extra offset arc is added to connect these two offset

segments. The radius of the added arc is the offset distance. The center of the added arc is the common end point of the two original segments. The starting and the ending angle of the added offset arc are related to the leading and the trailing offset segment separately. In the case of a CW closed loop, if the leading (or the trailing) segment is a line segment, the ending (or the starting) angle of the added offset arc is 90° lesser than the direction angle of the line segment. If the leading (or the trailing) segment is a CW arc, the ending (or the starting) angle of the added offset arc is 180° greater than the starting (or the ending) angle of the CW arc. If the leading (or the trailing) segment is a CCW arc, the ending (or the starting) angle of the added offset arc is the same as the ending (or the starting) angle of the CCW arc. Similar method can be applied if a CCW closed loop is involved.

5.1.2.B Case 2: the Angle Is Lesser Than 180°

First, the intersection between a pair of neighboring offset segments needs to be found. As shown in Figure 5.3, if an intersection exists, the intersection point will be assigned as the end point of this pair of offset segments. The assignment means that the data representing the offset segments need to be updated depending on the types of offset segment (line segment, CW arc, or CCW arc) and on the coordinate of the intersection point.

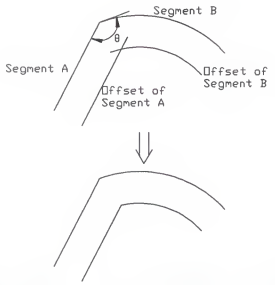


Figure 5.3 Connection of the intersecting offsets

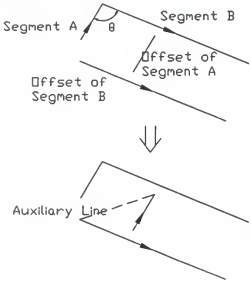


Figure 5.4 Adding auxiliary line while $\theta < 180^{\circ}$ and no intersection exists between two offsets

If no intersection exists between the pair of neighboring offset segments, as shown in Figure 5.4, an auxiliary line segment is added to connect the ending point of the trailing offset segment and the starting point of the leading offset segment. This auxiliary segment is marked with a different status from the ordinary offset segments. Later, the purpose of the auxiliary segment will be explained.

5.1.3 Splitting the Gross Offset Loop into Several Sub-loops

In many cases, the offset loop may result in the situation as shown in Figure 5.5 where the offset loop consists of five sub-loops. The sub-loops 1, 3, and 5 have the same orientation (CW) as the original loop, but not sub-loops 2 and 4 (CCW). The criteria of determining the validness of a sub-loop is by its orientation. When the sub-loops are in the opposite direction to their original loop, they are invalid. On the other hand, they are valid when they are in the same direction. Therefore, sub-loops 2 and 4 are discarded while sub-loops 1, 3, and 5 are kept. Notice that the computer is not capable of identifying whether a offset loop is a combination of many offset sub-loops or not. Thus, a procedure called "split" is created to divide the offset loop into several sub-loops if necessary. After that, the winding number calculation mentioned in Section 4.1 is used to determine the orientation of each sub-loop. The algorithm of "split" is as follows:

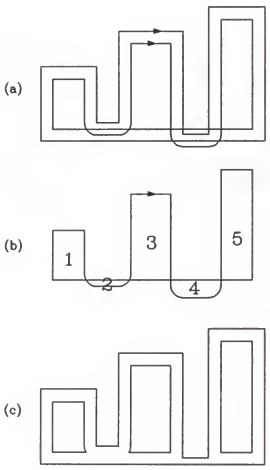


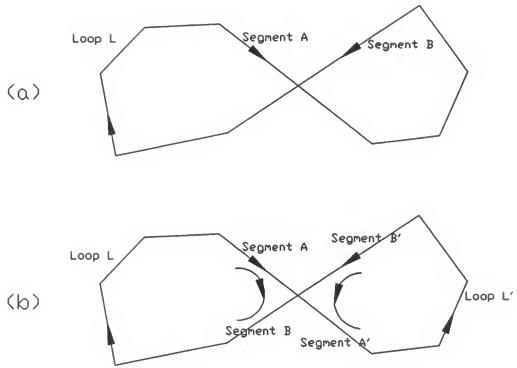
Figure 5.5 An example of using procedure "split" to result in multiple sub-loops

1. Initialize Loop L as the original offset loop.
2. Initialize Segment A as the starting segment of Loop L.
3. Initialize Segment B as the next segment of Segment A.
4. Find the intersection of Segment A and B. If no intersection exists, or only one exists while Segment A and B are two adjacent segments, update Segment B to its next segment.

If one or more intersecting points are found between Segment A and Segment B, a sibling of Loop L, say Loop L', is created by dividing Loop L into two loops at the intersecting point nearest to the starting point of Segment A.

The splitting procedure after the intersection point is found is shown in Figure 5.6. First, Segment A and B are divided into two segments from the intersecting point separately, say Segment A and A', and Segment B and B'. Memory blocks have to be allocated to save the data representing Segment A', Segment B', and Loop L'. Care must be taken to see if the intersecting point coincides with the ending points of Segment A (or B). In such case, no memory allocation is needed for Segment A' (or B'), and the next segment of Segment A (or B) is assigned as Segment A' (or B'). The second step is to reconstruct the linkages between segments and between loops. They are

```
Segment B'-> previous = Segment B -> previous;
Segment B' -> next = Segment A';
Segment A' -> next = Segment A -> next;
```




```
Segment B' -> previous = Segment B -> previous
Segment B' -> next = Segment A'
Segment A' -> next = Segment A ->next
Segment A' -> previous = Segment B'
Segment A -> next = Segment B
Segment B -> previous = Segment A
Loop L' -> next = Loop L -> next
Loop L -> next = Loop L'
Loop L' -> start = Segment B'
```

Figure 5.6 The "split" procedure after intersection point found

```
Segment A' -> previous = Segment B';
Segment A -> next = Segment B;
Segment B -> previous = Segment A;
Loop L' -> next = Loop L ->next;
Loop L -> next = Loop L';
Loop L' -> start = Segment B'.
```

Notice that the sequence of the above statements does matter. They are not allowed to switch randomly.

5. If Segment B is not the starting segment of Loop L, return to step 4.

6. Update Segment A to its next segment. If Segment A is not the previous segment of the starting segment of Loop L, go to step 4.

7. Update Loop L to its next sibling. If Loop L is not NULL, go to step 2.

8. Starting from the first sub-loop, every segment on each loop is checked for the existence of an auxiliary line segment. If one or more auxiliary line segments are on a sub-loop, this sub-loop is invalid. If the sub-loop contains no auxiliary line segments, the orientation of the sub-loop is verified. If the orientation is not the same as the original loop, the sub-loop is invalid too.

In term of programming techniques, each sub-loop is represented by a string of data which occupies an allocated memory block. When a sub-loop is declared invalid, this memory block has to be freed up so that it can be used again.

Otherwise, the memory block becomes an unused area in the system. For a pocket of complex geometry, while calculating the tool path, a lot of invalid sub-loops are created. Too many unfreed memory blocks may result in the system running out of memory and the calculations being terminated. Notice that before a sub-loop is abandoned, the linkages between its previous and its next siblings have to be set up.

5.2 Finding Offset Loops of Unequal Offset Distances

The process of finding the offset loops of unequal offset distances consists of three steps as follows:

1. finding the offset segment of each segment on the original loop, and then
2. connecting each pair of neighboring offsets to form a gross offset loop, and then
3. using the subroutine "split" of Section 5.1.3 to divide the gross offset loop into several sub-loops, and then using the "winding number" algorithm to check the legitimacy of each sub-loop.

The third step is the same as that used in finding the offset loops of equal offset distance. Since the procedure has been mentioned in Section 5.1, there is no need to explain it repeatedly. The first two steps are explained in more detail below.

5.2.1 Finding the Offset of Each Segment on the Original Loop

It has been explained in Section 1.2 that, when unequal offset distances are used, the offset segment of an arc will not be an arc. Therefore, a simple and preferred way of finding the offset of an arc is by using a series of line segments to approximate this arc, finding the offset of each line segment, and then connecting these offset line segments which become the offset of the arc. The criteria of approximating an arc by line segments is:

1. If the radius of the arc to be approximated is so small that an user specified allowable difference is greater than it, the arc is replaced by a line segment. Otherwise, the second and the third rule will be applied.
2. The interval of arc span for each line segment is no more than 20°.
3. The maximum difference between the arc and its approximated line segments can be no more than the allowable difference.

Figure 5.7 shows an example of the conversion. In the figure, D represents the user specified allowable difference, S the span of the arc, and R the radius of the arc. D, S, and R combined will determine the number of line segments, N, needed to approximate the arc. N, which is an integer, can be calculated by the following equation.

$$N = INTR\left(\frac{S}{2\cos^{-1}\left(\frac{R}{R-D}\right)}\right) + 1 \qquad (5.3)$$

As soon as N is found, the coordinates, $(P_{x,i}, P_{y,i})$ in Figure 5.7, of the end points of the line segments will be

$$(P_{x,i}, P_{y,i}) = (R\cos(\theta + \frac{i\,S}{N}), R\sin(\theta + \frac{i\,S}{N})) \quad for\ i=0,..,N \quad (5.4)$$

where θ is the starting angle of the arc. In order to save the data representing these line segments, memory allocation and linkage construction are needed.

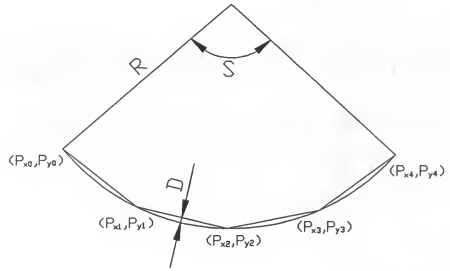


Figure 5.7 Approximation of arc by line segments

After every arc has been converted to a string of line segments, the only type of segment left on the original loop will be line segments. In addition, after the initial slotting, there are no outside corners left. Therefore, it is not necessary to add an extra arc between two offset segments. As a result, there will only be line segments involved in finding the offset loop of unequal offset distances.

5.2.2 Connecting Each Pair of Neighboring Offsets to Form a Gross Offset Loop

There are three cases which arise during the process of connecting the offsets of two neighboring line segments if unequal offset distances are used. The first case of connection occurs when the two offset segments intersect. In this situation, the connection is formed after the intersection point is set to be both the ending point of the preceding offset and the starting point of the trailing offset.

The offset segments do not cross each other in the second and the third case. These two cases can be distinguished mathematically by the following equation.

$$\theta = \theta_1 + \theta_2 - \theta_3 \qquad (5.5)$$

In this equation, θ_1 is the angle between the offset of the leading segment and an imaginary segment which connects the ending point of the leading offset segment and the starting point of the trailing offset segment. θ_2 is the angle between the imaginary segment and the trailing segment. θ_3 is the angle between the two original line segments. In the second case, the result of θ in Equation 5.5 will be π, and will be 3π in the third case.

The three cases are better demonstrated by Figure 5.8. Figure 5.8a is a closed loop formed by five line segments which are Segment A to Segment E. And Figure 5.8b shows the closed loop and the five offset segments, Segment A' to E', of

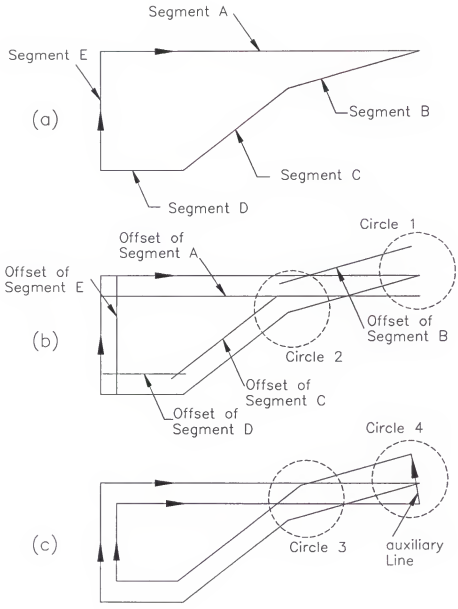


Figure 5.8 A example of finding offset loop of unequal offset distances
(a) The outer loop
(b) 5 offset segments found
(c) The gross offset loop found

Segment A to E separately. Since Segment E' intersects Segment A', the connection between these two offsets belongs to case 1. So are the connection between Segment D' and E', and between C' and D' as well. However, no intersection exists between Segment A' and Segment B', nor is there any between B' and C'. Figure 5.9 is a close-up look at circle 1 of Figure 5.8b. It shows the three angles, θ_1, θ_2, and θ_3, of Equation 5.5 between Segments A, B, and their offsets, A' and B'. Since the resulting θ is 3π, the connection between Segment A' and Segment B' is case 3. Figure 5.10 demonstrates the same angles between Segments B, C and their offsets at circle 2 of Figure 5.8b. The connection between B' and C' is case 2 since the resulting θ of Equation 5.5 is π.

The obvious solution for connection in the second case is shown in circle 3 of Figure 5.8c between the offset of Segment B and the offset of Segment C. The result can be achieved by
1. Finding the unlimited lines containing Segment B' and C'.
2. Finding the intersection point of these two unlimited lines.
3. reassigning the intersection point as the ending point of Segment B' and the starting point of Segment C'.

The right connection for the third case, shown in circle 4 of Figure 5.8c between the offset of Segment A and that of Segment B, is by appending an auxiliary line. The extra loop with the auxiliary line will be discarded later by executing the subroutine "split" mentioned in Section 5.1.3.

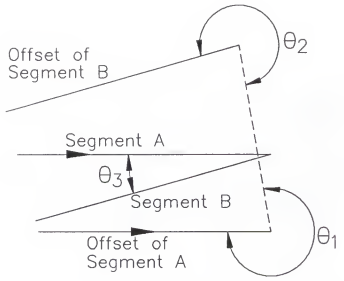


Figure 5.9 Case 3 connection: $\theta = \theta_2 + \theta_3 - \theta_1 = 540°$

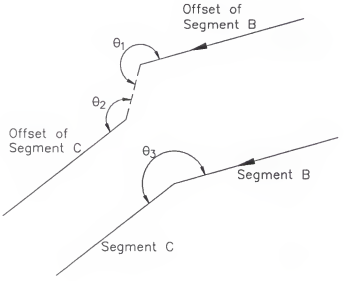


Figure 5.10 Case 2 connection: $\theta = \theta_2 + \theta_3 - \theta_1 = 180°$

Notice that in the situation as shown in Figure 5.11, the connection between the offsets of Segment A and B is a case 1 since they intersect with each other. However, since the offset of Segment B does not cross the offset of Segment C, the offset connection belongs to case 2 or case 3. By further calculation, θ of Equation 5.5 is 3π. It is thus categorized as a case 3 situation. An auxiliary line is needed to connect the two offsets of segments B and C. However, adding an auxiliary line directly as shown in Figure 5.11c would not achieve the desired outcome as shown in Figure 5.11e. As a result, in order to accommodate this kind of situation, if the offset finding is a case 3, before appending the auxiliary line, the preceding offset is extended backwards a certain distance, and the trailing offset forward the same distance. This certain distance is set to be the tool radius. The undesired loop, shown in the upper left part of Figure 5.11d, will be discarded later by executing "split" of Section 5.1.3.

5.3 Tool Path of a Pocket with Islands

When islands are present within the pocket boundary, the tool path finding procedure is more complicated than without islands. From the linked list depicting the island profiles (see Section 3.5 of island representation), another two linked lists with the same representation as that of the island profiles are created using the offsetting algorithm. Each node of the first created linked list represents the first

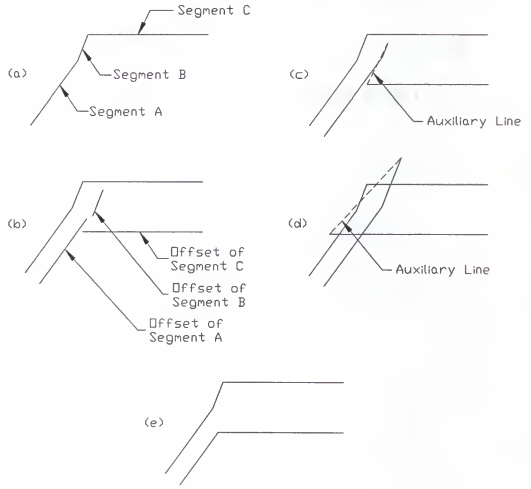


Figure 5.11 Another example of connecting offsets
(a) 3 segments on an outer loop
(b) 3 offset segments before connection
(c) Adding auxiliary line without extending the offsets of B and C
(d) Adding auxiliary line after extending the offsets of B and C
(e) The desired outcome

outer offset loop of an island profile. The offset distance is the same for all directions and is equal to the tool radius plus the radial depth of cut for finishing. Therefore, each node of the first linked list represents the slotting (contouring) path of an island profile and thus this linked list represents the slotting path of all of the island profiles. The nodes of the second linked list represent the outer offset loops of the contour tool path around the islands. The offset distances are the result of the optimization process (see Chapter 2) and may vary. The purpose of the second linked list, say "List outisland", is explained below.

After finding all sub-loops from a closed loop which is a descendent of the pocket profile, these sub-loops are checked to see if they are within, or intersect with any of the loops represented by the nodes of "List outisland". In other words, intersecting inspection is done between every segment on every sub-loop and each segment on the second expanded loop of every island profile. The fact that one or more intersection points exist between two segments proves that the two loops which the segments are on intersect with each other. The occurrence of an intersection point will trigger a simplified set Boolean operation: difference.

The difference operation is to find the outcome of area A minus area B, or A - B. The result is areas which are inside A, but outside B.

Figure 5.12 gives a better look at a situation when the difference operation is used. Beginning from the second shrunk loop of the pocket boundary, intersection is checked between the shrunk loop and the second expanded loop of the island profile. If no intersection exists, the shrinking process continues. As shown in Figure 5.12a, the process does not stop until an intersection is found between the fourth shrunk loop and the second expanded loop of the island profile. The desired outcome, shown in Figure 5.12c and 5.12d, is two closed loops. It is clear that if an area is represented by a closed loop, the difference between the two areas in Figure 5.12b will result in the two areas in Figure 5.12c. Although the concept of difference of areas has nothing to do with tool path generation, the results are found to match very well. Therefore, it is necessary to understand how the difference operation is functioning. Figure 5.13 shows a more detailed description of the above operation. In Figure 5.13a, Loop L_a and Loop L_b, are the same as those in Figure 5.12b. The detection of the first intersecting point, Point P, will trigger the difference operation. First, the orientation of Loop L_b is reversed. After that, Segment C and Segment D which intersect at Point P are divided into two segments separately. Segment C becomes Segment C and C', and D becomes D and D'. The linkages are then reassigned as follows:

```
Segment C' -> next = Segment D;
```

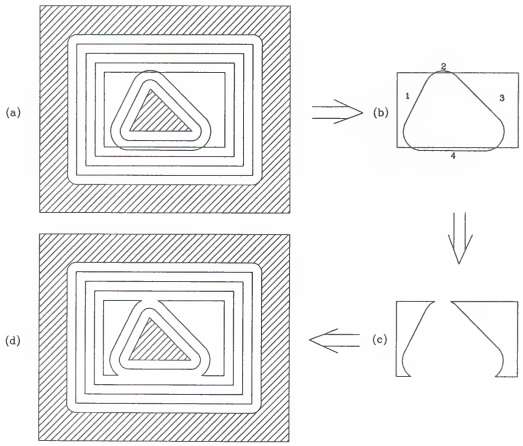


Figure 5.12 Resolution of crossing tool path resulting from a single loop by set operation difference

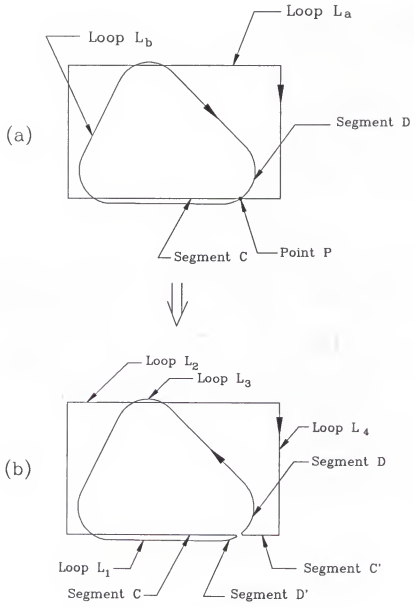


Figure 5.13 Detail of difference operation
(a) First intersection point, Point P, found
(b) Using the algorithm of "split" to perform difference operation

```
Segment D -> previous = Segment C';
Segment D' -> next = Segment C;
Segment C -> previous = Segment D'.
```

Assuming Point P is not an intersection point (which can be done by adding several "if" statements in a computer program), what is seen, instead of two intersecting closed loops, is a string of closed loops: Loop L_1, L_2, L_3, and L_4 as shown in Figure 5.13b. The algorithm of "split" mentioned in Section 5.1.3 will be used to sever these sub-loops. After that, the winding number (which indicates the orientation of a loop) of each sub-loop is calculated. Loops L_2 and L_4 are kept since their orientation is the same as Loop L_a which is the shrunk loop from the pocket boundary. On the other hand, Loops L_1 and L_3 are discarded.

Even if no intersection occurs, each sub-loop is investigated to find if it is within one of the loops which are the second expansion from the island profiles. The condition of a closed loop inside another closed loop has been explained in Section 4.1 of the winding number. If the condition is met, this sub-loop will be one of the innermost loops and will have no offset of its own. In other words, the tree node, representing this sub-loop, will be the last on the tree branch to which this tree node belongs. Figure 5.14 depicts the tool path of a pocket with a rectangular island. The second expanded loop from the island is portrayed as the dashed line loop. When the fifth shrunk loop from the pocket

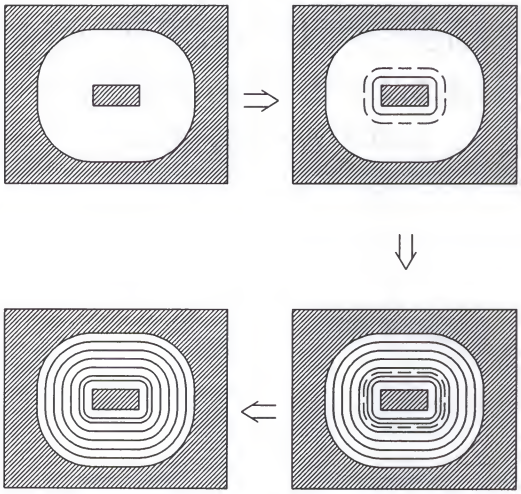


Figure 5.14 One case of loop termination when the shrunk loop from the pocket boundary profile is inside the second expanded loop of the island profile

profile is within the dashed line loop, this dashed line loop would be discarded, and the fifth shrunk loop would become the last of the spiral-in loops.

5.4 Loop Termination Conditions

Since the tool passes are closed loops obtained by continuously shrinking the pocket boundary, an important task is to determine when to terminate the shrinking process, or in other words, to determine which closed loops are the last ones. The last closed loop is either

1. a loop which can find no legal sub-loops, or
2. a loop which is inside the second expanded offset loop of an island profile.

The second condition has been demonstrated in Figure 5.14. As for a "legal sub-loop", it must satisfy both of the following conditions: a "legal sub-loop" must be

1. a closed loop which is in the same orientation as its original loop, and
2. a closed loop consisting of no auxiliary segments.

Figure 5.15 demonstrates two circumstances in which a loop terminates. In Figure 5.15a, the original loop is a clockwise triangle constructed by Segments A, B, and C. The offset loop is made by relimiting three offset segments of Segments A, B, and C. It results in a triangle of Segments A', B', and C'. Notice that, contrary to the original loop, the offset loop is CCW, and thus is not a "legal sub-loop". As shown in Figure

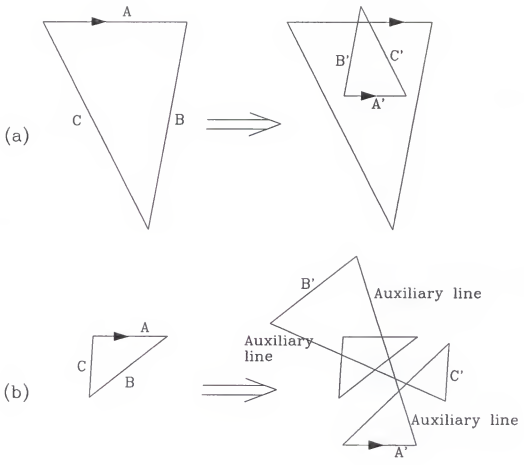


Figure 5.15 Loop termination conditions

5.15b, offset segments A', B' and C' do not intersect with each other. The further checking of θ's of Equation 5.5 verifies that the connections between A' and B', between B' and C', and between C' and A' are all case 3 scenarios. As a result, auxiliary lines are offered to form a link between the ending point of A' to the starting point of B', the ending point of B' to the starting point of C', and the ending point of C' to the starting point of A'. The "split" procedure will then result in four sub-loops. Since all of them contain at least one auxiliary line, they are void. Offset loop finding is terminated at the closed loop formed by Segments A, B, and C.

5.5 Connection between Larger Loops and Their Sub-loops

As mentioned in Section 3.3, the relationships between a pocket boundary and its tool pass loops form a general tree data structure in which each loop is a tree node. After each valid offset loop is found, the linkages with its outer loop and its sibling loop have to be constructed accordingly. The node representing the newly found loop is thus placed in the appropriate place in the tree. In the tree structure, the pocket boundary is the root node which is the highest rank tree node, and the outer tool pass loops are ranked ascendingly higher among the tree nodes. Those tool pass loops are the continuous offset loops from the pocket boundary obtained by using either equal or unequal offset distances.

After all the offset loops have been established, they need to be organized to become an integrated tool path. In other words, connection is essential between each larger loop and its sub-loops.

5.5.1 Two Cases of Connection between Loops

The way of linking is better demonstrated by Figure 5.16. Figure 5.16a displays a simple square pocket and its offset loops. Starting from the innermost loop, which is Loop 8 of Figure 5.16a, until the second successive offset loop of the pocket boundary, which is Loop 3, the starting segment of each loop is extended backward until this segment intersects with a segment on its outer loop. After that, the intersection point is assigned to be the starting point of the starting segment on the inner loop as well as to be the ending point of the intersected segment on the outer loop. Furthermore, this intersected segment is given a flag to symbolize it being the last segment of the outer loop. The result, shown in Figure 5.16c, becomes a true spiral tool path which is more efficient in terms of total distance of tool path compared to the conventional spiral tool path presented in Figure 1.7.

Topologically, the pocket boundary and its tool passes of the above case can be represented by a group of loops with the outermost loop symbolizing the pocket boundary as shown in Figure 5.17a. In the figure, Point 1 and Point 2 are the starting points of Loop C and Loop D separately. In figure

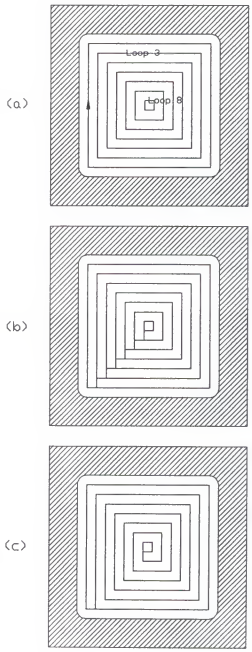


Figure 5.16 A simple case of connection of loops
(a) a square pocket and its tool pass loops
(b) starting segments are extended backward
(c) the intersected segments are trimmed

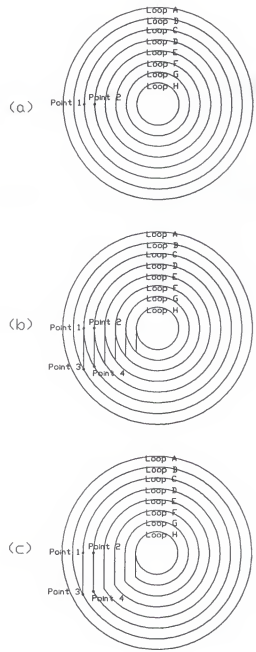


Figure 5.17 Topological view of connection of loops
(a) a group of loops
(b) loops are extended
(c) loops are trimmed to hinder the connection

5.17b, Loop C is extended backward from point 1 until it intersects Loop B at Point 3. So is Loop D from Point 2 until it intersects Loop C at point 4. The portion of Loop C between Point 1 and Point 4 is then trimmed as shown in Figure 5.17c. The outcome is that: C starts at Point 3 and ends at Point 4. It no longer resembles a closed loop. The same technique is used for Loops D, E, F, and G. Therefore, Loops C, D, E, F, and G are said to be "extended and trimmed". As for the innermost loop, Loop H, it is extended backward but not trimmed. Thus, it is said to be "extended only". Additionally, Point 3 is assigned as the starting point of Loop B.

The above case is too simplified to represent a general case, for each tree node, except the lowest rank one, has exactly one child. In more complicated cases, say the one shown in Figure 3.3, it is necessary to consider the loop connection in a more detailed fashion. In this case, the pocket boundary and its tool pass loops form a general tree as demonstrated by Figure 3.3b. Topologically these loops are represented by circles as in Figure 3.3a. To cross reference more conveniently, Figure 3.3b is moved to Figure 5.18a. The desired connection is demonstrated in Figure 5.18b. In the figure, the slotting pass starts from Point 1, around Loop B, and then back to Point 1. The spiral-in tool passes, starting at Point 1 right after the slotting is done, will follow the preorder tree traversal whose sequence, as mentioned in

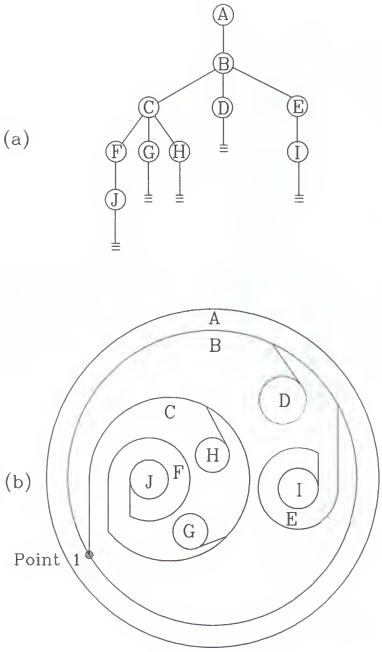


Figure 5.18 Loop connection on a more complicated case
(a) General tree representation of loops
(b) Connection finished

Section 3.6, is B → C → F → J → G → H → D → E → I. Notice that before the connection is achieved, each node represents a closed loop. After this connection, the closed loops will be either "extended and trimmed" or "extended only" according to the following rules:

1. The loops which enclose sub-loops are "extended and trimmed". Loops C, F, and E belong to this group in this case.

2. Those innermost loops are "extended only", such as Loops J, G, H, D, and I in this case. The innermost loop can be distinguished if the tree node, representing the loop, has a FIRST field which is null (see tree representation).

5.5.2 Retract and Approach

When traversing through a node which is not the first child of its parent, the cutter needs to be "retracted and approached" before entering the loop represented by this node. The meaning of "retract and approach" is:

1. moving the cutter in +Z direction a certain distance until it is above the top surface of the pocket, then

2. moving the cutter in X-Y plane to the starting position of the loop, and then

3. moving the cutter in -Z direction to the part surface.

If the following condition stands, A node is not the first child of its parent node.

```
a_node->parent->first != a_node,
```

where, != means not equal. For example, in the case of Figure 5.18, if a_node is Node G, it parent is Node C. The first child of Node C is Node F. Therefore, the left side is equal to Node F, which is not the same as the right side, Node G. Additionally, Node H, D, and E are not the first children of their parent nodes. In the case of Figure 5.18, the procedure of "retract and approach" is required while entering the loops represented by Nodes G, H, D, and E.

5.5.3 Change the Starting Segments of Loops

Since the connection of tool pass loops is made by extending backward the starting segment of each loop, it is necessary to ensure that the starting segment, say Segment A, of the loop which is the first child loop of its parent, is parallel to the starting segment, say Segment B, of its parent loop. In other words, Segment A should be originated from Segment B, or Segment A is the offset segment of Segment B.

Each offset segment contains a field called "origin". In the initial stages of finding offset loops, for each offset segment created, the field "origin" will be given the addresses of its original segment so that it is possible to trace back the origin of this segment later.

In the case shown by Figure 5.18, Node J is the first child of Node F, which is the first child of Node C. Therefore, the loops, represented by Nodes C, F, and J, should have parallel starting segments, and so do those tool pass

loops represented by Nodes E and I.

Every segment of an offset loop can trace its origin to the larger loop of this offset loop. However, not every segment on the larger loop has a descendent on its offset loop. If the starting segment, say Segment S, of a first child loop is not the offset of the starting segment of its parent loop, it is necessary to reassign the starting segment of the parent loop to be the origin of Segment S since this origin of S is always on the parent loop. Take the example of Figure 5.18, the starting segment of Loop F should be the origin of the starting segment of Loop J. If not, the starting segment of Loop F should be reassigned. The same procedure applies to Loop F and Loop C, and Loop I and Loop E. For Loop G, H, D, and E, since they are not the first children of their parent loops, the above procedure does not apply to them.

Since the redefinition of the starting segments of loops begins from the end of each tree branch, the preorder tree traversal, mentioned at Section 3.6, is used again to reach every node to determine whether it is the end of a tree branch.

5.5.4 Connection of Loops While Islands Exist

When islands exist inside the pocket boundary, the procedure of connection will be the same in most cases except that when all segments of any innermost loop can not trace

their origins back to the pocket boundary, but only to one of the island boundaries. Figure 5.19 shows an example of such a case. The pocket geometry and those tool passes before connection being made are shown in Figure 5.19a, and the corresponding binary tree representation is displayed in Figure 5.19b. Although Loop F is the first child of Loop E, all segments on Loop F are not originated from Loop E, but from the island loop. In this case, Loop E is considered as an innermost loop, although it is not. Loop C and D are "extended and trim, while Loop E is "extended only". The process of converting to a true spiral tool path will stop at Loop E. And then the passage from Loop E to Loop F is made possible by using the "retract and approach" method from Point 2 to Point 3. The final spiral starts at Loop F and ends at Loop G. The complete tool path is shown in Figure 5.19c where Point 1 is the starting position for both the slotting and spiral-in path.

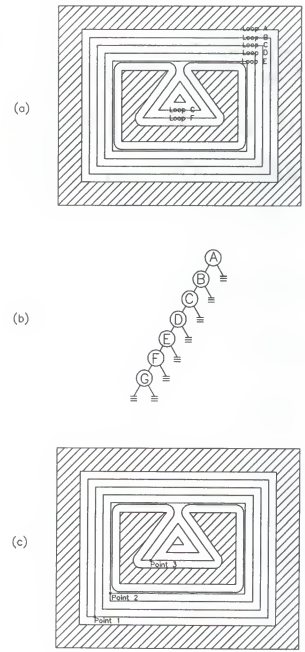


Figure 5.19 A case of loop connection when an island exists inside pocket boundary
(a) Tool pass loops before connection
(b) Tree representation of the tool pass loops
(c) Tool pass loops connected

CHAPTER 6

INSTALLATION PROCESSES

6.1 Installation of the New Pocketing Routine in CATIA

CATIA [13] by Dessault is installed on an IBM RT-10 workstation in the Machine Tool Laboratory at the University of Florida. One of the advantages of CATIA is the fact that it is a 3-dimensional CAD/CAM system while most other CAD/CAM commercial packages are either 2-dimensional or 2½-dimensional. Additionally, CATIA is a widely used package which provides complete access from the stage of designing to APT codes or CL data generation (explained in Section 6.2). Thus, it is by far more practical than some CAD/CAM softwares developed by most other researchers.

CATIA is capable of providing a utility that allows user written functions to be executed within the CATIA environment through the Interactive User Access (IUA) interpreted language. The IUA language provides several useful capabilities (options):

1) Select model elements using a digitizer to extract useful geometric data from the user designed model.

2) Access to the model's geometry for creation, modification, and deletion of geometric elements or NC operations (e.g. tool

paths generation) of the model.

3) Execution of subroutines developed in the FORTRAN language.

The IUA language is not suitable for complex numerical calculations. Thus, a better way to incorporate the new pocketing routine will be to use option 1 of the IUA to select desired elements from a drawing (such as the boundary and the bottom surface of a pocket), and then use option 2 to extract desired geometric data of these elements. The next step is to use option 3 to execute several FORTRAN subroutines which manipulate these data and generate data representing the tool paths. After that, option 2 is used again to define the NC tool paths according to these generated data. Notice that the main subroutines that will be incorporated with the IUA should be written in the FORTRAN language, but the supporting subroutines which are called by the main subroutines can be in either FORTRAN or C language.

An IUA procedure consists of three parts. The beginning of each part is indicated by a keyword. The first two parts are used to declare variables that the procedure will use. All variables that are required for the procedure execution must be declared.

The first part begins with the keyword INTERNAL. This part is used to declare the internal variables of the procedure and must exist even if the procedure uses no internal variables.

The second part beginning with the keyword EXTERNAL is used to declare external variables. Any variables authorized by means of the INPUT qualifier are external variables.

The third part begins with the keyword PROC and ends with the keyword END. This part consists of the executable part of the IUA procedure which is a sequence of executable instructions.

The following is an example of a CATIA IUA program which shows how the above three capabilities are used in IUA program.

```
INTERNAL
  CHAR*40 MSG INIT 'SELECT LINE OR FUNCTION KEY 'NO' TO EXIT'
  REAL    XCOORD, YCOORD, ZCOORD
EXTERNAL
  LN      LN1  INPUT
PROC
  SELECT   MSG LN1, NO
  IF (KODE EQ NO)  EXIT
  HIGHLT   LN1
  LOAD     EXTRACT  LN1, XCOORD, YCOORD, ZCOORD
  LOAD     MOVETO   XCOORD, YCOORD, ZCOORD
END
```

The first part of the procedure initializes the string variable MSG as 'SELECT LINE OR PRESS FUNCTION KEY NO TO EXIT', and declares variables XCOORD, YCOORD and ZCOORD as REAL. The second part declares variable LN1 as a geometric

entity LN (line).

The first statement of the third part will display the string variable MSG on the bottom section of the monitor screen, and will prompt the user to make a decision whether to select a line on the screen by the digitizer or to press the function key NO. If a line is selected, its address is stored in variable LN1. On the other hand, if the function key NO is pressed, the value representing NO is stored in the implicit variable KODE. The second statement checks the value of variable KODE. If its value is NO, then the procedure is interrupted by the EXIT instruction. The third statement will display the line which was selected by the digitizer in a highlighted mode. The fourth and fifth statements execute two FORTRAN subroutines named EXTRACT and MOVETO. Subroutine EXTRACT has arguments LN1, XCOORD, YCOORD, and ZCOORD. Its purpose is to extract the starting point coordinate from the selected line. Subroutine MOVETO, when invoked, will direct the cutting tool from the current position to the position defined by variables XCOORD, YCOORD, and ZCOORD.

The above procedure is a simple example. However, by using the same rules, it can be expanded such that all the geometric entities required to define a pocket are extracted and then manipulated to obtain desired geometric data, and then the cutter is directed by these data. The above approach is the principle of the new pocketing routine called "NEWPCKT". Figure 6.1 shows interactions between "NEWPCKT" and CATIA.

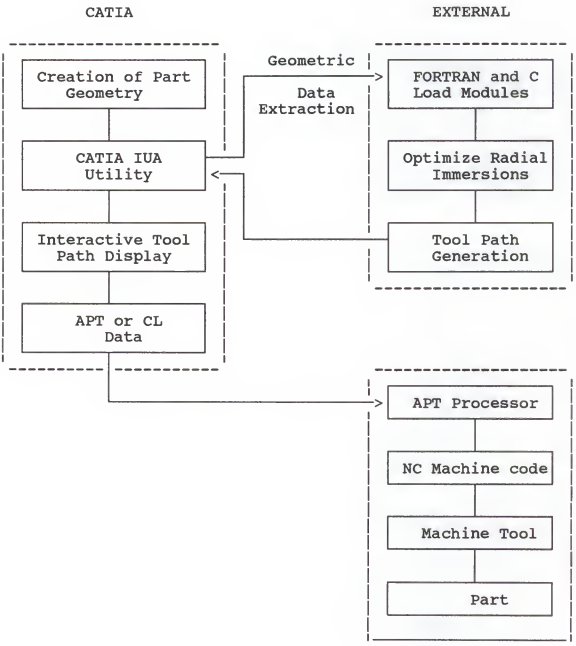


Figure 6.1 Interaction between CATIA and the new pocketing routine

6.2 APT and Machine Codes

APT (Automatically Programmed Tool) is a high level NC language that was designed to solve the following problems:

1. Defining the geometric entities used to specify a cutter path;
2. Calculating the cutter positions (coordinates) or path;
3. Defining the machining specifications needed for completing the machining process;
4. Outputing the cutter path and machining specifications in machine-understandable language (i.e., NC codes). [14]

The fourth task, however, requires an APT processor and a postprocessor.

Programming in APT depends, to a great extent, on the knowledge and experience of the programmer and on his spatial visualization of the part. [14]

CAD/CAM systems, widely used in industry for over 20 years to design, draw, store, and modify parts, ease the burden of defining machining process directly by APT codes. Most of those CAD/CAM systems use the geometric data of a CAD model to automatically generate a source machining program (e.g., in APT) and then use an existing NC processor and a suitable postprocessor to decode this APT program into machine-understandable NC code. CATIA is one of those systems that generates an APT part machining program from the geometric model. After the new chatter-free pocketing routine has been run in CATIA, an APT program defining the cutter path for

machining the pocket can be generated.

However, there is no APT program processor and postprocessor available in the Machine Tool Laboratory to decode the APT code generated by CATIA. An APT processor is therefore written although it can only accept limited APT commands which are used for the purpose of pocket machining. Those commands are

FEDRAT: define feed rate

LOADTL: load a cutter

FROM : define the starting position of cutter

GODLTA: move the cutter a distance of (dx, dy, dz)

GOTO : move the cutter to coordinate (x, y, z)

SPINDL: define the speed and the direction of rotation of the spindle

FINI : symbolize the end of the program

The method that the APT processor uses involves identifying the above commands and then generating corresponding NC codes. For NC codes, the word address format is the one used in modern CNC controller systems. Each word in a statement consists of a character identifying the meaning of the word and a number representing its content [14]. For example, the word specifying a tool position of Y = 1.50 is

Y1.50.

Some characters in the NC codes and their meaning are listed below:

N : statement label

S : spindle speed

X, Y, Z : coordinates in X, Y, and Z directions

M : operation of machine tool

G : code defining motion

T : cutter number

F : feed rate

Table 6.1 lists some examples of decoding APT codes into NC codes. In the table, some of the NC codes and their functions are explained below:

G90 : programming in absolute coordinates

G91 : programming in incremental coordinates

M06 : cutter change

M03 : spindle start clockwise

M04 : spindle start counterclockwise

M02 : end of program

Table 6.1 Examples of APT codes and their corresponding NC codes

APT CODES	NC codes
LOADTL / 1	N0001 M06 T01
SPINDL / 2000, CLW	N0006 M03 S2000
FEDRAT / 500.0 FROM / 0.0, 0.0, 0.0	N0011 G90 X0.0 Y0.0 Z0.0 F500.0
GOTO / 1.0, 1.0, 1.0	N0016 G90 X1.0 Y1.0 Z1.0
GODLTA /1.0, 2.5 1.1	N0021 G91 X1.0 Y2.5 Z1.1
FINI	N0026 M02

According to the method introduced above, a program called READAPT.C has been written to translate the APT program, which is generated by executing the chatter-free pocketing routine through CATIA, into an NC machine understandable program.

CHAPTER 7

EXAMPLES OF TOOL PATHS

Figure 7.1 shows the drawing of a rectangular pocket with two islands. The dimensions of the pocket are 450mm by 300mm by 2.286mm deep. To make a suitable comparison, two work pieces were machined by utilizing the tool paths from two different routines to produce the above pocket. One routine is the new chatter-free pocketing routine, and the other is the original routine from CATIA.

The tool used to machine this pocket was a 25.4mm by 107.0mm 4-flute HSS end mill mounted in a set-screw type tool holder on the White-Sundstrand machining center in the Machine Tool Laboratory. According to the database, shown in Table A.2 of Appendix A, the spindle speed was set at 2720 rpm, and the feed rate at 1360 mm/min.

When the chatter-free routine was used, the optimization process, mentioned in Chapter 2, determined that 5 layers were needed for the initial slotting and 2 layers were needed for the spiral-in machining to reach the entire depth of the pocket. Therefore, the optimum axial depth of cut for the initial slotting was 0.4572mm and that for spiral-in machining was 1.143mm. The optimum radial immersions for spiral-in were

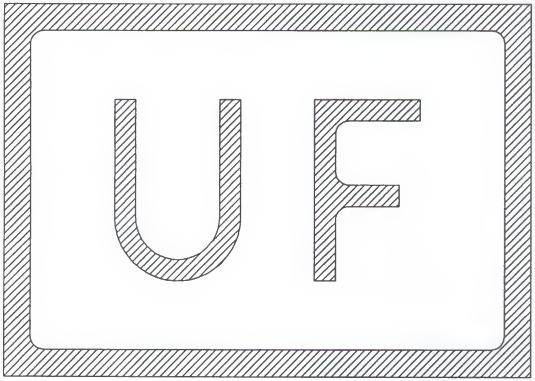


Figure 7.1 The drawing of the sample pocket

then determined from the database and the optimum axial immersion. Figure 7.2 shows the tool path generated by this routine. In the figure, the first inner offset loop from the pocket boundary and the first outer offset loops from the island boundaries mark the tool passes for slotting. The dashed lines in the figure represent the passes when the tool travels in air. All other lines in the figure represent the spiral-in tool passes. The total machining time was 20.30 minutes.

Figure 7.3 shows the tool path generated by the original pocketing routine from CATIA to machine the same pocket using the same tool. The tool path was a spiral-out starting at Point 1. The maximum radial immersion, required to run this routine, was set at 70%. From Table A.2 of Appendix A, the corresponding maximum allowable axial immersion is 0.78mm for the radial immersion of 70%. Therefore, to achieve the depth of 2.286mm, 3 layers were required, with each layer being 0.762mm deep. Since the machining time was calculated as 9.10 minutes for each layer, the total machining time for the 3 layers was 27.30 minutes.

The tool path from the original routine in CATIA was a spiral-out, starting as a slotting pass at the middle portion of the pocket and then machining outward. Therefore, although the maximum radial immersion was set to be 70%, at the slotting portion and at corners which were inevitable for the spiral-out option, the radial immersion exceeded 70% which

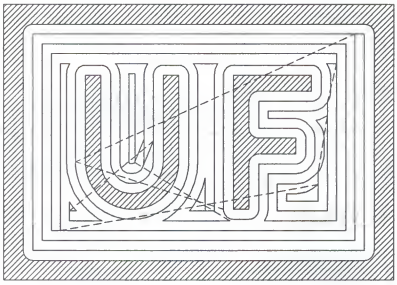


Figure 7.2 The tool path generated from the new chatter-free routine

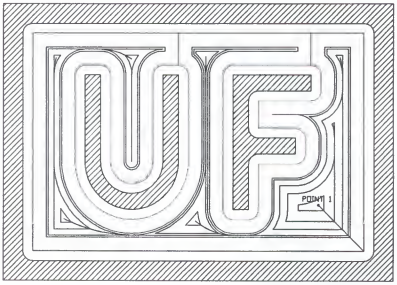


Figure 7.3 The tool path generated from the original routine of CATIA

caused chatter. Figure 7.4 shows the sound signal when chatter occurred during the machining of the sample pocket using the routine from CATIA. Figure 7.5 shows the spectrum of the signal. The chatter frequency is 1210 Hz. Figure 7.6 shows the sound signal taken when machining the sample pocket using the chatter-free routine. Figure 7.7 shows the spectrum of the sound signal. It is obvious that the magnitude is small compared to the signal of Figure 7.5.

Comparing the machining times between the new chatter-free routine and the original routine from CATIA, it is seen that the former one is 34% more efficient than the latter one. Additionally, to generate a chatter-free tool path from the original routine, the axial immersion is forced to be reduced from 0.762mm to 0.4572mm, the maximum allowable axial immersion for full radial immersion (refer to Table A.2 of Appendix A). The machining time becomes 45.5 minutes which is 2.24 times longer than the machining time encountered when using the tool path resulting from the new chatter-free routine.

Figures 7.8 and 7.9 show the photographs of the finished parts produced by both the original routine from CATIA and the chatter-free routine. Notice that the chatter marks are very distinct in Figure 7.8. On the contrary, only feed marks are visible in Figure 7.9. Figures 7.10 and 7.11 show two more examples of tool paths for exotic pockets that the new routine is capable of generating.

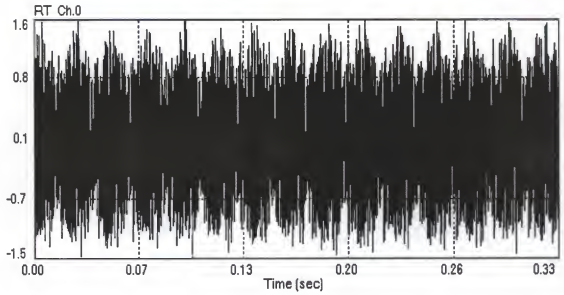


Figure 7.4 Sound signal taken when chatter occurred during the machining of the sample pocket using the routine provided by CATIA

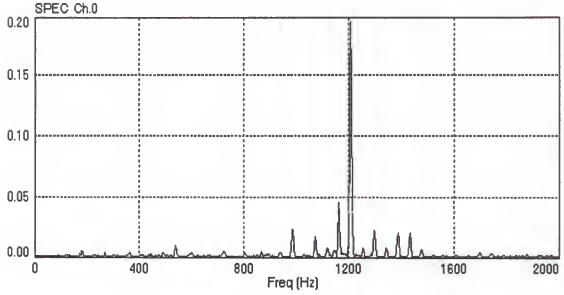


Figure 7.5 The spectrum of the above sound signal

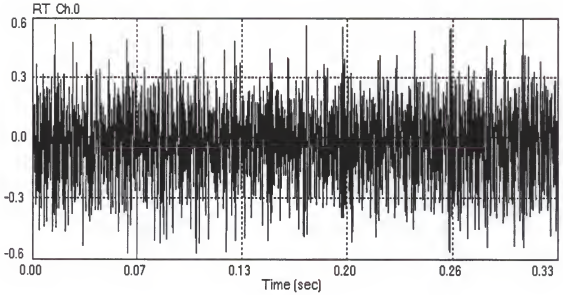


Figure 7.6 Sound signal taken when machining the sample pocket using the chatter-free routine

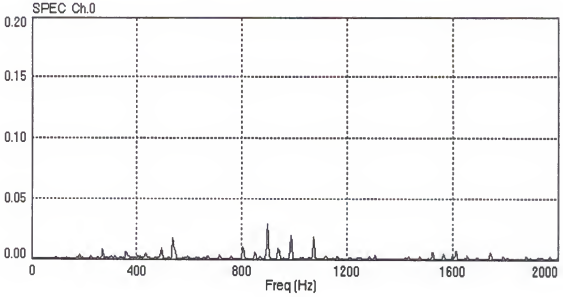


Figure 7.7 The spectrum of the above sound signal

(a)

(b)

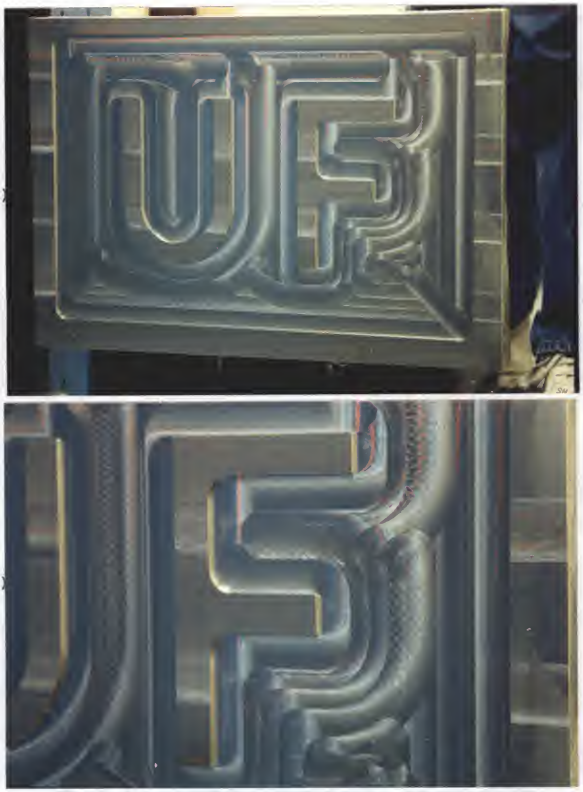


Figure 7.8 (a) The finished part machined by the tool path from the original pocketing routine from CATIA
(b) A closeup look at the chatter marks

(a)

(b)

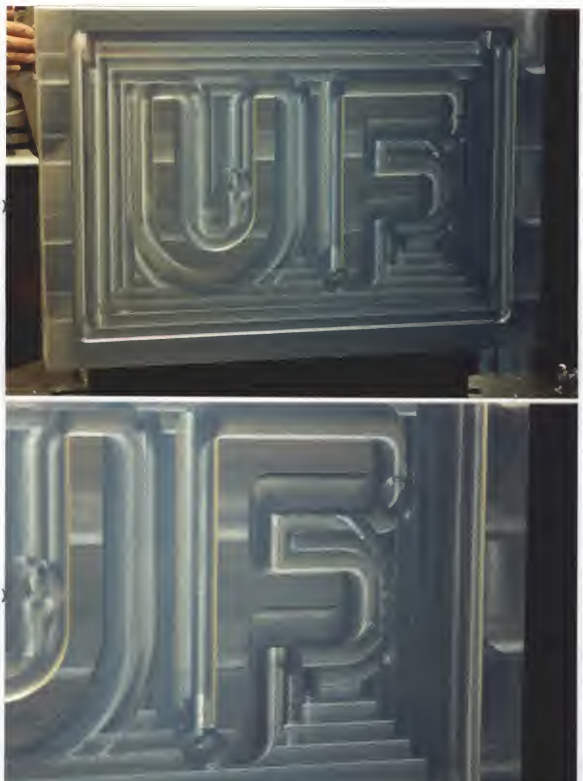


Figure 7.9 (a) The finished part machined by the tool path from the new chatter-free pocketing routine
(b) A closeup look at the feed marks

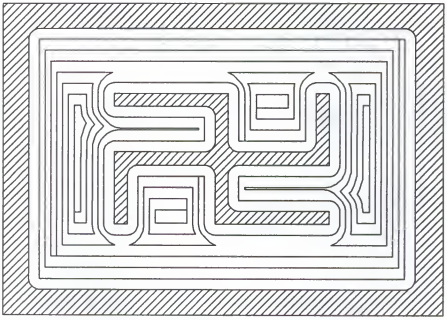


Figure 7.10 An example of tool path

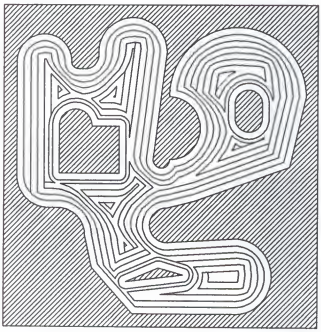


Figure 7.11 An example of tool path

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

The study described in this dissertation concerns the development of a new pocketing routine in the field of NC tool path generation. This new routine would meet certain quality criteria by using appropriate axial and radial depths of cut selected from a database of acceptable cuts which has been filled either through simulation or through experimentation. While the criteria in this case is the avoidance of chatter, the required database can also be prepared using any other criterion such as surface finish, accuracy, cutting force, etc, or using a combination of criteria.

The optimum chatter-free pocket machining is achieved by the combination of the following strategies:

1. An optimization process, which utilized the database in association with the geometry of the pocket, is used to automatically determine the optimum radial and axial immersions for the pocket machining in terms of minimizing the production time. The optimized radial immersions might vary for different machining directions. As a result, the routine is capable of allowing the radial immersion to vary.
2. Tool paths are designed spirally from outside in and are

designed to follow the arc shapes around the outside corners. The strategy is to avoid abrupt cutting angle increases at the corner positions which may cause chatter.

The pocketing routine has been significantly improved by including the possibility of multiple islands and by using true spiral tool paths. The basic implementation has been in CATIA. Additionally, a processor for APT programs was developed to generate NC machine codes.

Work pieces have been cut using the tool paths generated by the new pocketing routine as well as by the original routine of CATIA. The former tool path showed about 34% higher metal removal rate than the latter one. It should be noted that this was under the assumption that the programmer using the conventional software (either CATIA or others) was aware of the dynamic capabilities of the NC machines (which is not generally the case now) and therefore the time for verification cuts and rewriting, typically needed during the programming stage was not included in evaluating the improvement. By comparison to common practice now, the saving achieved by using the new routines would be dramatically larger.

Although the new routine provides a better way for machining the pocket, it is limited in the design of tool paths for pockets with flat bottom surfaces. A good expansion to the new routine will be: using the same strategies mentioned above to create better tool paths for pockets with

3-D sculptured surface than those created by the original routine of CATIA.

Another approach that can make further improvement to the routine in terms of machining time will be by using the geometry modeling techniques to verify the tool paths. The tool paths are designed to achieve chatter-free machining which means that the cutting angles would be equal to the angles implied by the radial immersions. However, in many places along the tool paths such as around inside corners or narrow necks of the pockets, especially with pockets of complex geometry, the cutting angles are actually lesser or even zero. In such occasions, a higher feed rate than the specified one is feasible. The geometry modeling techniques provide such an instrument to obtain the cutting angles at any location along the tool paths. This data could then be used to determine an increment ratio of the feed rate. As a result, the machining time could be reduced.

APPENDIX A

THE CHARACTERISTIC OF A CUTTING TOOL

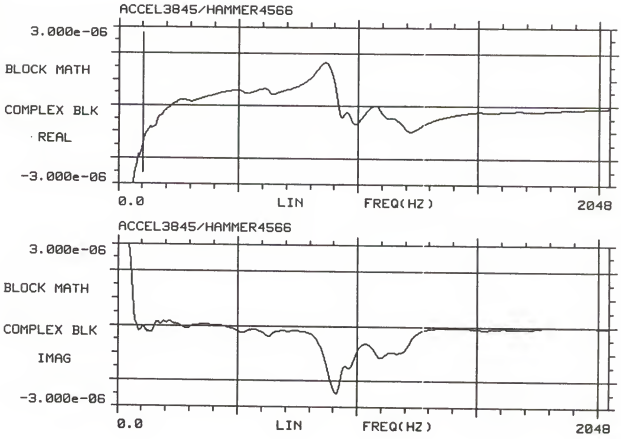


Figure A.1 The X direction transfer function for a 25mm by 107mm 4-flute HSS end mill cutter on White-Sundstrand machining center

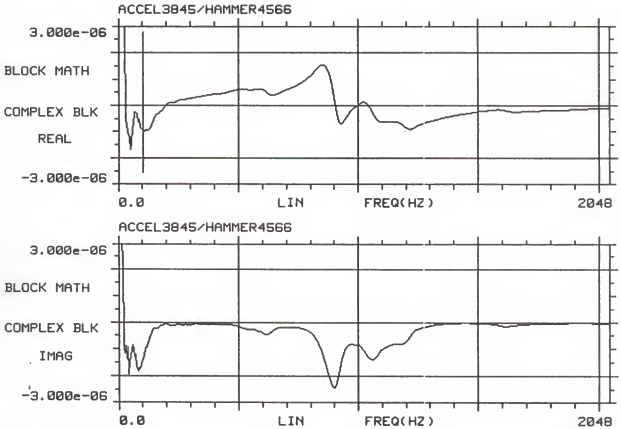


Figure A.2 The Y direction transfer function for a 25mm by 107mm 4-flute HSS end mill cutter on White-Sundstrand machining center

Table A.1 Parameters of a 25mm by 107mm 4-flute HSS end mill cutter on White-Sundstrand machining center

Mode		Natural Freq (HZ)	Stiffness (N/m)	Mass (kg)	Damping ratio
X dir	1	910	5.149e6	0.157	0.039
	2	1093	2.160e7	0.38	0.021
	3	1160	1.790e7	0.34	0.031
Y dir	1	900	4.980e6	0.156	0.041
	2	1061	1.240e7	0.279	0.031
	3	1190	3.100e7	0.55	0.023

Table A.2 Permissible depths of cut for a 25mm by 107mm 4-flute HSS end mill cutter on White-Sundstrand machining center

			Axial Immersion for Four Cutting Dir. (mm)			
			+X	+X+Y	+Y	-X+Y
Radial Immersion	Up Milling	0.2	1.423	1.642	1.511	1.435
		0.3	1.317	1.552	1.432	1.326
		0.4	1.142	1.383	1.201	1.179
		0.5	1.037	0.9886	1.168	1.0395
		0.6	0.965	0.8668	0.946	0.931
		0.7	0.882	0.747	0.793	0.8505
		0.8	0.756	0.693	0.747	0.819
		0.9	0.630	0.567	0.567	0.6615
		1.0	0.4725	0.504	0.504	0.4725
	Down Milling	1.0	0.4725	0.504	0.504	0.4725
		0.9	0.630	0.567	0.567	0.6615
		0.8	0.756	0.693	0.7875	0.819
		0.7	0.882	0.7875	0.882	0.8505
		0.6	1.071	1.008	1.0395	0.945
		0.5	1.134	1.071	1.071	1.0395
		0.4	1.260	1.2915	1.26	1.197
		0.3	1.2915	1.575	1.2915	1.3545
		0.2	1.512	1.827	1.3545	1.512

* Spindle speed at 2720 rpm, and feed rate at 1360 mm/min.

BIBLIOGRAPHY

1. Kshonze, S., "Dynamic Pocketing Routines for NC Machining," Master's Thesis, University of Florida, 1989.

2. Tlusty, J., Smith, K. S., and Zamudio, C. A., "New NC Routines for Quality in Milling," Annals of the CIRP, Vol. 39, pp 517-521, 1990.

3. Persson, H., "NC Machining of Arbitrarily Shaped Pockets," Computer-Aided Design, Vol. 10, No 3, pp 169-174, 1978.

4. Suh, Y. S. and Lee, K., "NC Milling Tool Path Generation for Arbitrary Pockets Defined by Sculptured Surface," Computer-Aided Design, Vol. 22, No 5, pp 273-284, 1990.

5. Tiller, W. and Hanson E. G., "Offsets of Two-Dimensional Profiles," Computer-Aided Design, Vol. 4, No 9, pp 36-46, 1984.

6. Pham, B., "Offset Approximation of Uniform B-Spline," Computer-Aided Design, Vol. 20, No 8, pp 471-474, 1988.

7. Hoschek, J., "Offset Curves in the Plane," Computer-Aided Design, Vol. 17, No 2, pp 77-82, 1985.

8. Coquillart, S., "Computing Offsets of B-Spline Curves," Computer-Aided Design, Vol. 19, No 6, pp 305-309, 1987.

9. Tlusty, J., "Machine Dynamics," Handbook of High Speed Machining Technology, King, R. I., editor, Chapman and Hall, New York, 1985.

10. Tsai, M. D., Takata, S., and Inui, M., "Operation Planning Based on Cutting Process Models," Annals of the CIRP, Vol. 40, pp 95-98, 1991.

11. McCracken, D. D., and Salmon, W. I., A Second Course in Computer Science with MODULA-2, John Wiley, New York, 1987.

12. Apostol, T. M., Calculus II, Blaisdell Pub. Co., Waltham, Mass., 1967.

13. CATIA User's Manual, IBM Corp., Kingston, New York, 1988.

14. Chang, C. H., and Melkanoff, M. A., NC Machine Programming and Software Design, Prentice Hall, New york, 1989.

BIOGRAPHICAL SKETCH

E-Hsung Cheng was born in Taipei, Taiwan on January 7, 1959. He earned his Bachelor of Science degree in Mechanical Engineering from National Taiwan University in June 1981. In May 1987, he was granted a Master of Science degree from the University of Florida. In August of 1987 he entered the Ph.D program in the Department of Mechanical Engineering of the University of Florida, and has been conducting research under the direction of Dr. Jiri Tlusty since January of 1988.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jiri Tlusty, Chair
Graduate Research Professor
of Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Donald W. Dareing
Professor of Mechanical
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Senser Yeralan
Associate Professor of
Industrial and Systems
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John K. Schueller
Associate Professor of
Mechanical Engineering

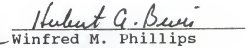
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Kevin Scott Smith
Assistant Professor of
Mechanical Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

MAY, 1992



for Winfred M. Phillips
Dean, College of
Engineering

Madelyn M. Lockhart
Dean, Graduate School